# CONVEX Loader
# User's Guide

Document No. 740-002430-204

Fifth Edition
December 1989

CONVEX Computer Corporation
Richardson, Texas USA

*CONVEX Loader*
*User's Guide*
Order No. DSW-011
Fifth Edition



Printed in the United States of America

# Revision Information for
# *CONVEX Loader User's Guide*

| Edition | Document No. | Description |
|---|---|---|
| Fifth | 740-002430-204 | Released with ConvexOS V8.0 in December 1989. Includes editorial and formatting changes plus the following:<br><br>Chapter 1<br>• Adds iterative loading example<br><br>Chapter 2<br>• Updates loading C and FORTRAN programs<br><br>Chapter 4<br>• Updates optional header flags<br><br>Chapter 5<br>• Adds standard system libraries<br><br>Appendixes<br>• Adds error and warning messages |
| Fourth | 740-002430-203 | Released with V7.0 of the operating system in October 1988. |
| 3.0 | 740-002430-201 | Released with V6.1 of the operating system in October 1987. |
| 2.0 | 740-000230-000 | Released with V2.0 of the operating system in September 1985. |
| 1.0 | 740-000230-000 | Initially released with V1.0 of the operating system in February 1985. |

# Table of Contents

# Appendices



# List of Tables



# List of Figures

# Preface

## Purpose and Audience

The *CONVEX Loader User's Guide* provides general information about the loader and specific details about available options. This guide:

- Provides pertinent facts about the CONVEX loader, including the *ld* command and its options
- Illustrates and explains how to load C and FORTRAN programs manually
- Describes the object file format used for version V6.0, and earlier, of the operating system
- Describes the standard object file format (SOFF) used for version V6.1, and later, of the operating system
- Discusses object libraries

The *CONVEX Loader User's Guide* addresses experienced programmers wishing to use the loader for specific applications.

You need no particular experience with loaders to use this manual. If you are unfamiliar with programming on the CONVEX system, consult the associated documents listed in this preface.

## Organization

This guide is organized into the following chapters and appendixes:

- Chapter 1 provides a conceptual overview of loader operations and functions.
- Chapter 2 describes the format of the loader command *ld* and describes options and file arguments used with the command.
- Chapter 3 describes the object file format for systems running V6.0, or earlier, of the operating system.
- Chapter 4 describes the SOFF for systems running V6.1, or later, of the operating system.
- Chapter 5 provides information on location and contents of standard object file libraries.
- Chapter 6 provides programming examples.
- Appendix A provides descriptions of some ambiguous error messages.
- Appendix B provides a sample load map.
- Appendix C provides instructions for reporting software or documentation problems to the CONVEX Technical Assistance Center (TAC).

## Notational Conventions

The following conventions are used in this document:

- Words enclosed in rounded rectangles indicate keyboard keys that you press. For example, (RETURN) refers to the carriage return key. Words separated by a hyphen and enclosed in rounded rectangles indicate two keys that you must press simultaneously. For example, (CTRL-X) indicates that you must press the (CTRL) key while simultaneously pressing the keyboard ⊗ character key.

- **Boldface** type indicates user-entered information for a computer program. You should enter these command sequences exactly as they appear.

- Brackets ( [ ] ) designate optional entries. Brackets are also used to distinguish parts of command strings that may be omitted.

- A horizontal ellipsis ( . . . ) shows repetition of the preceding item(s).

- A vertical ellipsis shows continuation of a sequence where not all of the statements in an example are shown.

- Commands, utilities, and files that are documented in the *ConvexOS Programmer's Reference* are italicized; occurrences that include a number enclosed in parentheses refer to the appropriate section of the manual. For example, *vmstat*(1) means that documentation for the *vmstat* command is located in Section 1.

- The | symbol is used in command sequences in which you must pick no more than one alternative from a list of command options. For example, in the following command sequence,

  (fp)> **s[et] s[pu-selftest]** = [*d[isable]* | *e[nable]*]

  you must choose either *d[isable]* or *e[nable]*, but not both.

- The percent symbol ( % ) signifies the standard C shell user prompt.

## Associated Documents

Using this guide successfully may require information not specific to the tasks described herein or not within the scope of this guide. The following documents may answer questions to help you solve problems encountered when working through this book. CONVEX Computer Corporation provides the following related documents.

- *ConvexOS Programmer's Reference.* This guide documents the ConvexOS operating system, an implementation of the Berkeley UNIX operating system containing POSIX functionality with extensions for supercomputing environments.

- *CONVEX Architecture Reference.* This manual describes the architecture and instruction set used by the CONVEX computer system.

- *CONVEX Assembly-Language User's Guide.* This guide is for experienced programmers who wish to develop assembly-language programs for the CONVEX system.

- *CONVEX C User's Guide.* This guide describes the CONVEX C compiler and supplements Kernighan and Ritchie's *C Programming Language.*

- *CONVEX FORTRAN Language Reference Manual.* This reference manual defines the CONVEX extensions to the FORTRAN-77 standard and supplements the *CONVEX FORTRAN User's Guide.*

- *CONVEX FORTRAN User's Guide.* This guide describes how to compile, link, debug, and execute FORTRAN programs.

- *CONVEX UNIX Primer* provides an introduction to ConvexOS. Recommended for novice users, it describes how to log in and start using the CONVEX system and explains many of the commonly used commands, the *vi* text editor, and the C shell.

## Ordering Documentation

To order the current edition of this or any other CONVEX document, you need to know the exact title or the six-character order number. Refer to the copyright page of this document for its six-character order number. To find the order numbers for other CONVEX documents, refer to the *CONVEX COMPUTER Price Book* or call the Technical Assistance Center or your local CONVEX office.

To order an edition other than the current edition, you need to know the 12-digit document number. Refer to the title page of this document for its 12-digit document number. To find document numbers for other CONVEX documents, call the Technical Assistance Center or your local CONVEX office.

To order CONVEX documentation, send requests to

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, TX 75083-3851 USA

## Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC). Use the phone numbers in the following table.

| Location | Phone Number |
|---|---|
| Within the continental U.S. | 1(800)952-0379 |
| Outside continental U.S. | Contact local CONVEX office |

# Chapter 1
# Conceptual Overview

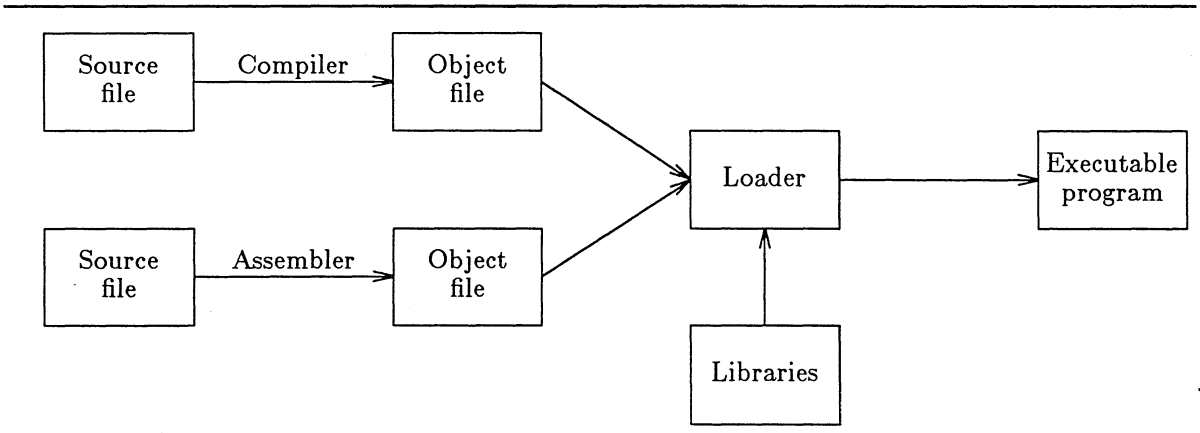
## 1.1 Introduction

In general, a loader is a program that prepares the output of language processors for execution. The CONVEX loader, *ld*, performs this function by:

- Combining separately compiled object modules to produce executable programs
- Resolving symbolic references between modules
- Searching libraries of previously compiled object modules to resolve symbolic references

Loaders enable you to use separately compiled subroutines and object archives housing libraries of previously compiled subroutines. These capabilities support modular programming and enable you to use the machine more efficiently. Loaders are sometimes called *linkers* or *link editors* because they resolve external reference linkage between separately compiled modules. Figure 1-1 illustrates the basic functions of the loader.

**Figure 1–1: Loader Functions**




The CONVEX loader does not load an executable program into main memory; programs are loaded into main memory when they are executed.

## 1.2 Loader Input

Only two types of files, object files and library files, are used as input to *ld*. An object file is a single object module that contains binary instructions and data. Library files contain frequently used object modules that the loader inserts into programs.

Each object file produced by the loader contains a symbol table. The */usr/include/a.out.h* file defines the layout of the symbol table entries. Symbols and the symbol table for the object file format used by the operating systems through V6.0 are discussed in Chapter 3; Chapter 4

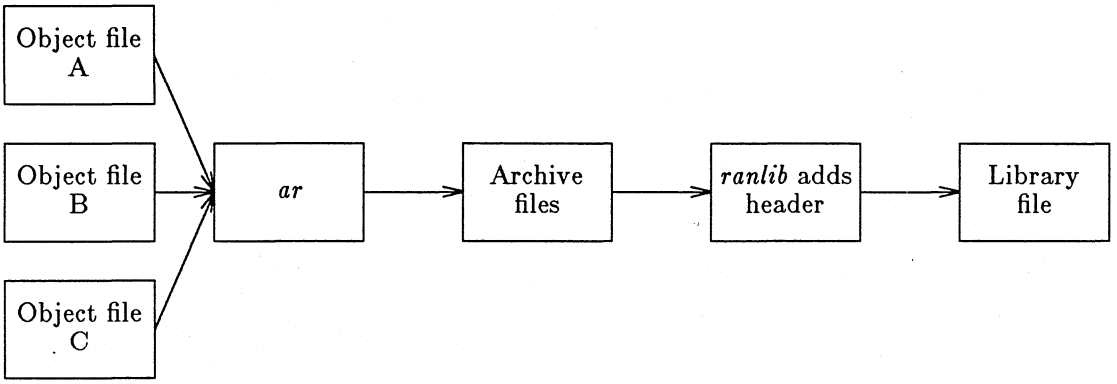
discusses the SOFF produced by operating systems V6.1 and later.

To produce library files, follow these steps:

1. Merge object files into archive files with the *ar* program.
2. Create a library header for the archive file using the *ranlib* program; *ranlib* adds header information to the beginning of the archive files, which enables the loader to quickly locate symbols in the file.

Figure 1-2 illustrates this procedure. Object-file libraries are also discussed in Chapter 5.

**Figure 1–2: Generating Library Files**




## 1.3 Loader Output

Object files produced by *ld* have the same internal format as loader input object files. For *ld*, the standard output is a file called *a.out*. The loader makes this file executable (by setting the file's execute-permission bits) if there are no errors or unresolved references in the input. Chapter 3 discusses the format of the object files produced by operating systems through V6.0. Chapter 4 discusses the SOFF produced by operating system V6.1 and later.

## 1.4 Loader Functions

This section provides an overview of the basic loader functions. These functions include

- Linking object files
- Resolving internal references
- Resolving external references
- Searching libraries
- Iterative loading

### 1.4.1 Linking Object Files

The four stages in the linking process are:

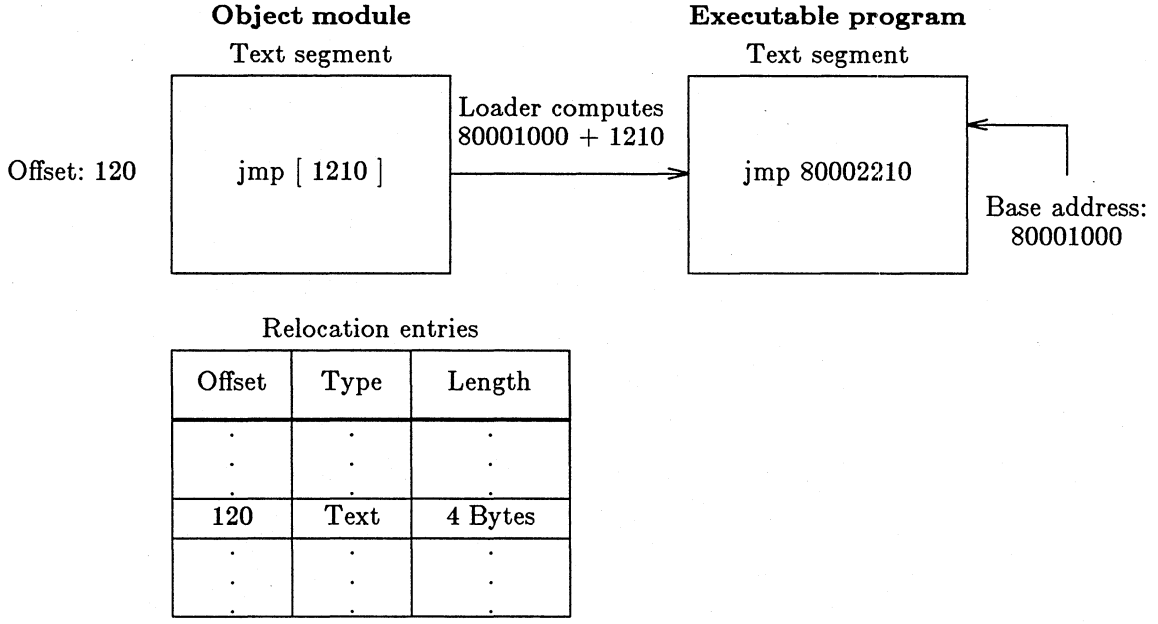
1. The loader resolves address references in the program.
2. The loader resolves any references made in the file to symbolic names in other object files or library files.
3. The loader creates an executable program file (unless you specify the *-r* option).
4. The loader produces an optional "load map" of symbols and references and reports unresolved external references and other errors.

Not all instructions in the CONVEX assembly-language instruction set run on every machine in the CONVEX family of supercomputers. Each assembly-language instruction is tagged within the assembler to be of a certain type. This *type* corresponds to the machine(s) that will accept the instruction. The types of all instructions in one assembled file are stored in the flags-word of the executable file's header. The loader accumulates all of the flags-words from each object file and stores them in the flags-word in the final executable file. No loader option to ignore these instruction types currently exists.

### 1.4.2 Resolving Internal References

The loader translates addresses of any relocatable data within object modules into absolute addresses in main memory for the generated executable program. Within each object module, the loader corrects machine instructions that reference main memory by adding a bias factor to account for the final location of the referenced data within main memory. Within the object module, the assembler generates memory references as offsets within the *.text .data, .tdata, .bss,* or *.tbss* segments of the program. The assembler also adds a special relocation record to the generated object file that instructs the loader to resolve that address by adding the base address of the appropriate segment. The loader must fill in the actual address of the data. It does this by adding the base address of the appropriate segment (*.text, .data, .tdata, .bss* or *.tbss*) to the offset stored in the instruction and corrects the instruction in the generated executable program to reference the correct final address. Figure 1-3 illustrates this action.

**Figure 1–3: Resolving Internal References**




## 1.4.3 Resolving External References and Library Searches

The loader makes two passes through the object files and libraries specified on the *ld* command line to resolve external references and perform library searches. In the first pass, the loader looks at the symbol table of each object file to find symbols that resolve references. If a symbol remains unresolved after this procedure, the loader looks in the header of subsequent libraries to find a symbol in that library that resolves the reference.

In the second pass, the loader "patches" locations containing symbol references with physical symbol addresses. As a second step, the loader copies from the library any object modules that were "marked" on the first pass as resolving references. The loader also generates an output file.

## 1.4.4 Iterative Loading

The loader can be used to combine small object files into a larger object file using a technique called iterative loading. To perform iterative loading, use the *-r* option on the command line.

Iterative loading minimizes overhead associated with reading and processing many smaller object modules. You can combine stable routines into a larger object file, and leave undeveloped or changing routines accessible. Decreasing the number of object files can reduce the time required to link executable programs.

The following simple example of iterative loading uses *a.c*, *b.c*, and *c.c* as sources.

```
% cat a.c
 main()
 {
    f();
 }

% cat b.c
 f()
 {
    g();
 }

% cat c.c
 g()
 {
    printf("hello\n");
 }
```

1. Compile all the sources to objects:

```
% cc -c {a,b,c}.c
  a.c:
  b.c:
  c.c:
```

2. Load *b.o* and *c.o* into a common object file:

```
% ld -r -o ofiles b.o c.o
```

3. Load *a.o* with the object file containing *b* and *c*:

```
% cc a.o ofiles
```

4. Execute your program:

```
% a.out
  hello
```

# Chapter 2
# *ld* Command

## 2.1 Introduction

This chapter describes the format of the *ld* command and provides information on the options and file arguments used by the command.

> **CAUTION** The loader should be automatically invoked by a compiler. If you manually invoke the loader, loading order and library names might change from one software release to the next. Loader command lines in this document are given only as examples.

## 2.2 Command Format

The *ld* command invokes the loader program. You can use this command either interactively or as part of a shell script. The format of the *ld* command is

**ld** [*options*] [*objfiles*] [*libraries*]

The *options* you can specify on the command line are described in section 2.4, "Options." By convention, names of *objfiles* end with ".o" and names of *libraries* end with ".a".

Arguments follow *ld* on the command line. The loader processes the argument files in the order specified. Libraries should generally appear at the end of the command line in order to resolve external references.

Following are examples of typical *ld* commands. The command sequence

```
ld -o mumble foo.o bar.o
```

combines the object files *foo.o* and *bar.o* into the executable program file, *mumble*.

The command sequence

```
ld -o myprog /lib/crt0.o main.o subr1.o subr2.o -lc
```

combines the C-generated object files *main.o*, *subr1.o*, and *subr2.o* with the C runtime initialization routine *crt0.o* and the C runtime library. The program file is named *myprog*.

The sequence

```
ld -o myprog /lib/crt0.o main.o subr1.o subr2.o -lU77 -lF77 -lI77 -lD77 -lm -lc
```

combines the FORTRAN-generated object files *main.o*, *subr1.o*, and *subr2.o* with the FORTRAN runtime initialization routine *crt0.o* and the FORTRAN libraries. The program file is named *myprog*.

## 2.3 Loader Processing

Following is a general description of the processing carried out by the loader.

1. Each object file encountered on the command line is unconditionally loaded. As the object file is encountered, symbols are added to the global symbol table and segment sizes are accumulated.

2. If a library is encountered, it is scanned and objects from it are loaded *only* if they satisfy an unresolved external reference. If they do, they are processed just like object files encountered.

3. If unresolved symbols remain and the *-m* option was specified, the libraries are rescanned until the load is complete or no symbols are resolved.

4. After the first pass through the input files, the common blocks are allocated space at the end of the *.bss* and *.data* sections. The special symbols (*_end, _etext, _edata, _etdata,* etc.) are then created. *_end* is always created; the others are created if referenced.

5. Next, values of all symbols in the symbol table are adjusted by origins of their respective segments.

6. Each of the included object files and library members are now reprocessed in the order they appeared on the command line. For each object module, the symbol table is processed to check for multiply-defined symbols, and the symbols are written to the output file. The text and data for the object are then copied to the output file, with relocatable references properly updated.

## 2.4 Options

Options available with the loader are given below. You can list options that have no following values on the command line either separately (e.g., *-s -d*) or in a concatenated form (e.g., *-sd*). Options are preceded by a hyphen. With the exception of *-fmode, -lname,* and *-Moption,* options and arguments can be separated by a space or not, as you prefer.

The order in which the options *-l, -t, -u, -y, -L,* and *-NL* are arranged on the command line is significant. This is because these options are interpreted when they are encountered as the command line is scanned on the first pass of the loader. Also, ordering of flags *-s, -x,* and *-X* is extremely important. These options *must* precede any object files on the command line. If they are encountered after any objects or libraries are encountered, an error message is printed and the load aborts. Refer to Appendix A for descriptions of some ambiguous error messages. If you use the *-s, -x,* or *-X* options, place them first on the command line. The order of the other loader options is not significant. The following options are available:

**-D** *size* Instructs the loader to add zeros to the data segment until it matches the specified *size* (a hexadecimal number).

**-d** Instructs the loader to force a common storage definition even if the *-r* option is present.

**-E** *mode* Changes the execution *mode* of an existing executable image. You can specify more than one option on a command line, setting the options specified and clearing all others. This option understands the following modes: *-E demand,* which sets the demand-paged option; *-E prepage,* which sets the prepaged option; *-E nonswap,* which sets the nonswapped option; and *-E posix,* which sets the POSIX executable option (for use with higher-level languages only).

**-e** *name* Instructs the loader to treat *name* as the symbol name that will be the entry point of the loaded program. Location *start* is the default. If *start* is not defined, the loader uses *_main*. If *_main* is undefined, the loader uses the base of the text segment (this causes a warning message to be issued).

**-F** Forces creation of an *a.out* executable file even if undefined externals still exist when the load operation finishes. The *a.out* file produced does not have the executable-mode bit set; you must use *chmod*(1) to make the file executable. This option also causes any out-of-date libraries to have *ranlib*(1) executed on them. Normally, the load simply terminates.

**-f***mode* Specifies which mode the image produced should run in. The *-fi* option specifies IEEE mode; object files can be IEEE. The *-fn* option specifies native mode; object files can be native.

> **Note** Your system administrator could have chosen a default mode (either native or IEEE). Check to see if a default mode exists and it is the one you want to use.

**-H** *xxxx* Leaves a hole of size *xxxx* (hex) at the end of the text segment.

**-L** *path* Searches this path for libraries (in addition to the regular search path). You can use this option more than once; each use adds a path to the search list. Paths specified by *-L* are searched before the standard library paths.

**-l***name* Is an abbreviation for the library name */lib/libname.a*, where *name* is a string. If that library does not exist, *ld* searches for */usr/lib/libname.a*. If that library in turn does not exist, *ld* searches for */usr/local/lib/libname.a*. The loader searches a library when it encounters the library's name, so the placement of the option is significant.

**-M***option* Instructs the loader to produce a load map. If no *option* is given, a primitive map consisting of a list of the object files included is produced. Three predefined maps are available. The *option S* produces a short map, *M* produces a medium map, and *L* produces a long map. You can also create your own map by specifying just the sections you want output. You can select from the following sections:

| | |
|---|---|
| *h* | Map header |
| *l* | Library modules included |
| *o* | Object modules included |
| *g* | Global symbols sorted by address |
| *s* | Global symbols sorted by name |
| *n* | Local symbols for each object file by name |

The sections *h, l,* and *o* make up the short map, sections *h, l, o, g,* and *s* make up the medium map, and all the sections together make up the long map.

You can also use *option P* with any of the predefined maps or when specifying individual sections. *P* causes most of the formatting to be removed from the map so that it can be more easily processed by other programs. Refer to Appendix B for a sample load map.

**-m** Performs multiple scans of libraries. If any unresolved externals remain at the end of the load operation, the libraries specified on the command line are rescanned. Libraries are scanned in the order specified on the command line and the scanning continues until one full pass is made and no object modules are extracted.

**-NL** Ignores the standard search path for libraries. The option *-NL* in conjunction with the *-L* feature enables you to scan only the directories you specify.

**-o** *file* Instructs the loader to use *file*, instead of *a.out*, as the name of the *ld* output file.

**-R** *ring* Relocates the resulting output file to the ring number specified in the *ring* argument. Typically, this option is used during the system-generation procedure to rebuild the operating system residing in ring 0. Because of the protection mechanisms in operation, it is impossible to execute a program that specifies a ring number less than 4 while the system is running. The default is ring 4. Refer to the *CONVEX Architecture Reference* for more information.

**-r** Instructs the loader to generate relocation bits in the output file so that the file can be loaded again. This option also prevents final definitions from being given to common symbols and suppresses the "undefined symbol" diagnostics.

**-s** Saves space by "stripping" the output of the symbol table and relocation bits. (One disadvantage to this process is that it impairs the usefulness of the debuggers.) This information can also be removed from the output file via the *strip*(1) command.

**-T** *origin* Signals the loader to use *origin* (a hexadecimal number) as the address of the text segment origin. The default origin is 1000 hex (4096 decimal). Any origin you specify must be a multiple of 4096 (decimal).

**-t** Causes the loader to print the path name of each file as it is processed.

**-u** *sym* Causes the loader to create *sym* as a symbol and enter it as undefined in the symbol table. This capability is useful for loading routines exclusively from a library, because initially, the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. Refer to section 2.8.5, "Loading FORTRAN Programs From Libraries Only," for an example.

**-v** *ver* Sets the version stamp in the file header. Same as executing the *vers*(1) command with the *-v* option.

**-w** Suppresses warning messages from the loader.

**-X** Used by the C and FORTRAN compilers to discard internally generated labels (those that begin with *L*, *.L*, or *ABS* symbols) while retaining symbols local to routines.

**-x** Instructs the loader to omit local (nonglobal) symbols from the output symbol table and only enter external (global) symbols. This option saves some space in the output file.

**-y** *sym* Displays each file in which the symbol *sym* appears. This option also displays the symbol type and indicates whether the file defines or references *sym*. For example, if the loader reports that a symbol (e.g., *_ralph*) is unresolved, the load can be run again by specifying the option *-y _ralph*. The loader responds with a list of names of all object files that refer to *_ralph*. (It is usually necessary to begin *sym* with an underscore "_" because external C and FORTRAN variables begin with underscores.)

## 2.5 IEEE Consistency Checking

To support IEEE floating-point hardware, the loader uses two floating-point format options to control the type of objects that you can validly load together and to determine the execution mode of the resulting executable file. Table 2-1 identifies each option, the acceptable object-file modes, and execution mode of the image produced.

### Table 2-1: Options Specified by User

| Option | Acceptable Object-File Modes | Executable Mode |
|---|---|---|
| -fn | Native mode | Native |
| -fi | IEEE mode | IEEE |

The loader checks the floating-point mode bits for each input file to make sure that the combination of objects is valid under the specified option. If the floating-point mode of the object files does not agree with what you specified on the command line, a message similar to the following is displayed (this is the message for NATIVE; IEEE is similar):

```
ld: NATIVE mode objects may not be loaded if -fi is specified
    /lib/crt0.o                    - pre-SOFF (assumed native)
    foo.o                          - pre-SOFF (assumed native)
ld: ERROR - Invalid floating point modes encountered
ld: No executable produced
```

**Note**

Files */lib/crt0.o* and *foo.o* are examples only. File names that appear here depend on the names of your files.

If you do not specify an option, the loader determines one for you. Table 2-2 indicates what happens when the loader encounters object files where the floating-point format is not specified.

### Table 2-2: No Options Specified by User

| Object Types Processed | Executable Mode | Description |
|---|---|---|
| Native mode only | Native | Only native mode was encountered so the mode of *a.out* is native. |
| IEEE mode only | IEEE | Only IEEE mode was encountered so the mode of *a.out* is IEEE. |

## 2.6 Reprocessing Executable Files

If you give the loader a single executable file as input, it uses the options you have specified to set the flags in the file header and create a new executable file with the appropriate bits set. For example, a normal executable file can be converted to a nonswapped executable without having to go through the entire load process a second time, or the floating-point format option for an executable file can be changed to indicate IEEE format.

> **Note**
>
> This type of conversion can be performed on only one executable file at a time; it is an error to specify more than one executable file as input to the loader (no object files can be specified).

Table 2-3 indicates options that can be used for reprocessing executable files, along with the action performed when the option is used.

**Table 2-3: Reprocessing Files**

| Option | Action |
|---|---|
| -fn | Sets native mode. |
| -fi | Sets IEEE mode. |
| -E demand | Sets the demand-paged option. |
| -E prepage | Sets the prepaged option. |
| -E nonswap | Sets the nonswapped option. |

## 2.7 Loading C Programs

The following sections discuss:

- How to load C programs manually
- How to load profiled C programs
- How to load C programs for use with *csd*
- How to load programs that call math functions

### 2.7.1 Loading C Programs Manually

Loader processes should be initiated automatically by a compiler. The loader can be invoked manually, however, to load previously assembled files. To invoke the loader manually for C programs, examples of loader commands follow:

**ld -fi /lib/crt0.o /usr/lib/fpmode_i.o** *objfiles* **-lc**

**ld -fn /lib/crt0.o** *objfiles* **-lc**

where:

| | |
|---|---|
| **-fi** | Accepts IEEE-mode object files. |
| **-fn** | Accepts native-mode object files. |
| **/lib/crt0.o** | Is the C runtime initializer that sets up the environment your program needs to run (*/lib/crt0.o* must be the first object file loaded). |
| **/usr/lib/fpmode_i.o** | Is the file that actually defines your execution mode as IEEE; it sets the variable *_mth$fpmode* that tells *crt0* to set the IEEE bit in the Processor Status Word (PSW) and check for IEEE hardware (always use with *-fi*). |
| *objfiles* | Are the object files to be loaded. |
| **-lc** | Is the option that instructs the loader to scan */lib/libc.a* (the C library). |

Command-line options that you select depend on which format you used to compile/assemble your object files. If you decide to omit the *-fmode* option and the *fpmode* path name from the *ld* command line, it defaults to either *-fi* or *-fn*, depending on the mode of the object files (refer to section 2.5, "IEEE Consistency Checking").

## 2.7.2 Loading Profiled C Programs

To profile a C program using the *prof*(1) profiler, the following example command could be used:

**ld -f***mode* **/lib/mcrt0.o** *objfiles* **-lc_p**

where:

| | |
|---|---|
| **/lib/mcrt0.o** | Is substituted for the regular initialization routine. |
| **-lc_p** | Is the profiled version of the C library. |

To profile the program using the *gprof*(1) profiler, the following example command could be used:

**ld -f***mode* **/usr/lib/gcrt0.o** *objfiles* **-lc_p**

where:

| | |
|---|---|
| **/usr/lib/gcrt0.o** | Is substituted for the regular initialization routine. |

## 2.7.3 Loading C Programs for Use With *csd*

To use the source-code debugger *csd*(1), the following example command could be used:

**ld -f***mode* **/lib/crt0.o** *objfiles* **-lg -lc**

where:

**-lg** Is the option that instructs the loader to scan */usr/lib/libg.a* (an object file that *csd* uses).

### 2.7.4 Loading C Programs That Call Math Functions

The following example shows how to load a C program that calls math functions:

**ld -f***mode* **/lib/crt0.o** *objfiles* **-lm -lc**

where:

**-lm** Instructs the loader to scan */usr/lib/libm.a* (a math library).

To load a profiled C program that calls math functions, the following command sequence shows an example:

**ld -f***mode* **/lib/mcrt0.o** *objfiles* **-lm_p -lc_p**

where:

**-lm_p** Is the profiled version of the math library.

The following command sequence illustrates how to use the *gprof*(1) profiler:

**ld -f***mode* **/usr/lib/gcrt0.o** *objfiles* **-lm_p -lc_p**

To use the source-code debugger *csd*(1) on a C program that calls math functions, the loader could be invoked with the following example command:

**ld -f***mode* **/lib/crt0.o** *objfiles* **-lm -lg -lc**

## 2.8 Loading FORTRAN Programs

The following sections discuss:

- How to load FORTRAN programs manually
- How to load profiled FORTRAN programs
- How to load FORTRAN programs compiled with *-vfc*
- How to load FORTRAN programs for use with *csd*
- How to load FORTRAN programs from libraries only

### 2.8.1 Loading FORTRAN Programs Manually

To invoke the loader manually for FORTRAN programs, the following commands show examples:

**ld -X -fi /lib/crt0.o /usr/lib/fpmode_i.o** *objfiles* **-lU77 -lF77 -lI77 -lD77 -lm -lmathC[1 | 2] -lc**

**ld -X -fn /lib/crt0.o** *objfiles* **-lU77 -lF77 -lI77 -lD77 -lm -lmathC[1 | 2] -lc**

where:

| | |
|---|---|
| **-X** | Discards internally generated labels (must be placed first on the command line). |
| **-fi** | Accepts IEEE-mode object files. |
| **-fn** | Accepts native-mode object files. |
| **/lib/crt0.o** | Is the FORTRAN runtime initializer that sets up the environment your program needs to run (*/lib/crt0.o* must be the first object file loaded). |
| **/usr/lib/fpmode_i.o** | Is the file that actually defines your execution mode as IEEE; it sets the variable *_mth$fpmode* that tells *crt0* to set the IEEE bit in the PSW and check for IEEE hardware (always use with *-fi*). |
| *objfiles* | Are the object files to be loaded. |
| **-lU77** | Instructs the loader to scan */usr/lib/libU77.a* (the FORTRAN utility library). |
| **-lF77** | Instructs the loader to scan */usr/lib/libF77.a* (the FORTRAN intrinsic library). |
| **-lI77** | Instructs the loader to scan */usr/lib/libI77.a* (the FORTRAN I/O library). |
| **-lD77** | Instructs the loader to scan */usr/lib/libD77.a* (the dummy VMS-to-UNIX file name translation library). |
| **-lmathC[1 \| 2]** | Instructs the loader to scan either the C1 or C2 math intrinsic function library. |

## 2.8.2 Loading Profiled FORTRAN Programs

To profile a FORTRAN program using the *prof*(1) profiler, the following command shows an example:

**ld -X -f***mode* **/lib/mcrt0.o** *objfiles* **-lU77_p -lF77_p -lI77_p -lD77 -lm_p -lmathC[1 | 2]_p -lc_p**

where:

| | |
|---|---|
| **-lU77_p** | Is the profiled version of the FORTRAN utility library. |
| **-lF77_p** | Is the profiled version of the FORTRAN intrinsic library. |
| **-lI77_p** | Is the profiled version of the FORTRAN I/O library. |
| **-lmathC[1 \| 2]_p** | Is the profiled version of either the C1 or C2 math intrinsic function library. |

To profile the program using the *gprof*(1) profiler, the following command shows an example:

**ld -X -f***mode* **/usr/lib/gcrt0.o** *objfiles* **-lU77_p -lF77_p -lI77_p -lD77 -lm_p -lmathC[1 | 2]_p -lc_p**

## 2.8.3 Loading FORTRAN Programs Compiled With *-vfc*

To load FORTRAN programs that have been compiled with the VAX FORTRAN compatibility flag (*-vfc*), the following command could be used:

**ld -X -f***mode* **-o***file* **/lib/crt0.o** *objfiles* **-lV77 -lU77 -lF77 -lI77 -lD77 -lm -lmathC[1 | 2] -lc**

where:

| | |
|---|---|
| *file* | Instructs the loader to use *file* as the output file. |
| **-lV77** | Instructs the loader to scan */usr/lib/libV77.a* (the VAX FORTRAN compatibility constants library). |

## 2.8.4 Loading FORTRAN Programs for Use With *csd*

To use the source-code debugger *csd*(1), the loader could be invoked with the following command:

**ld -X -f***mode* **/lib/crt0.o** *objfiles* **-lU77 -lF77 -lI77 -lD77 -lm -lg -lmathC[1 | 2] -lc**

## 2.8.5 Loading FORTRAN Programs From Libraries Only

It is possible to load from libraries only. In other words, you can perform a load without specifying any *.o* files on the *ld* command line. Use the *-u* option, as illustrated in the following example command (remember, */lib/crt0.o* must be the first object file loaded):

**ld -X -f***mode* **/lib/crt0.o -u _MAIN__ sublib.a -lU77 -lF77 -lI77 -lD77 -lm -lmathC[1 | 2] -lc**

In this example, the loader is instructed to generate an undefined external by the name of *_MAIN__*. This is the CONVEX FORTRAN internal routine that gets control at runtime and eventually invokes the user's main program. The *-u* switch causes the loader to start searching the libraries to resolve the undefined external reference to the symbol *_MAIN__*. This starts the process that ends with the loading of all required modules.

# Chapter 3
# Object File Format

## 3.1 Introduction

This chapter describes the seven sections of the object file: header, text segment, data segment, text-relocation information, data-relocation information, symbol table, and string table. This file format is used for object files produced with the operating systems through V6.0. Starting with V6.1 of the operating system, CONVEX uses a different file format, called SOFF, described in Chapter 4 of this manual.

## 3.2 Header Format

The header contains a number of entries that provide information on the file format and the size and location of various segments of the file. Figure 3-1 shows the format of the header.

**Figure 3–1: Header Format**

```
/*
 * Header prepended to each a.out file
 */
struct exec {
        long                    a_magic;        /* magic number */
        unsigned long           a_text;         /* text segment size */
        unsigned long           a_data;         /* initialized data size */
        unsigned long           a_bss;          /* uninitialized data size */
        unsigned long           a_syms;         /* size of symbol table */
        unsigned long           a_entry;        /* entry point */
        unsigned long           a_trsize;       /* size of text relocation */
        unsigned long           a_drsize;       /* size of data relocation */
        unsigned long           a_talign;       /* byte alignment of text */
        unsigned long           a_dalign;       /* byte alignment of data */
        unsigned long           a_balign;       /* byte alignment of bss */
        unsigned long           a_torigin;      /* address of start of text segment */

};

#define OMAGIC  0507    /* unaligned assembler output */
#define NMAGIC  0510    /* compromise format */
#define ZMAGIC  0513    /* demand load format */
#define PMAGIC  0515    /* prepaged load format */
#define LMAGIC  0517    /* prepaged, nonswapped load format */
```

where:

| | |
|---|---|
| *a_magic* | First 32 bits of the header file, which specify the object file type. If the magic number is 0513 (octal), the data and text segments in the object file are aligned on 4096-byte boundaries, enabling demand-paging of the file. These files are called *ZMAGIC*-format files. If the magic number is 0507 (octal), the file is unaligned and unexecutable and is called an *OMAGIC*-format file. |
| *a_text* | Size of the program text segment, in bytes. |
| *a_data* | Size of the initialized portion of the data segment, in bytes. |
| *a_bss* | Size of the uninitialized portion of the data segment, in bytes. |
| *a_syms* | Size of the symbol table, in bytes. |
| *a_entry* | Address at which program execution begins. |
| *a_trsize* | Size of the text relocation commands, in bytes. |
| *a_drsize* | Size of the data relocation commands, in bytes. |

## 3.3 Text Segment

The text segment of an object module contains the executable binary code and read-only constants. The text segment of object files with the *ZMAGIC* magic number begins at byte address 4096 in the object file. This 4096-byte alignment allows the operating system to demand-page the program directly from the executable file. *ZMAGIC* format files have their text and data segments padded to 4096-byte boundaries. *OMAGIC* format files begin the text segment immediately after the header. (The *N_TXOFF* macro in *a.out.h* returns the absolute position of text in *OMAGIC* format files when given the name of an *exec* structure as an argument.)

## 3.4 Data Segment

In *ZMAGIC* format files, initialized data starts at the next 4096-byte boundary after the end of the text segment. In *OMAGIC* format files, initialized data immediately follows the text segment.

## 3.5 Relocation Entries

Certain portions of any given object module may contain references to addresses whose values are not known when the object file is assembled. For example, instructions in the text segment of the object module may refer to addresses in the data segment. These data segment addresses must be relocated (biased) by the final memory base address of the data segment. In cases like these, the assembler communicates these unresolved addresses to the loader via relocation entries (see Figure 3-2), and the loader generates the final addresses of all the unknown values.

There are two sets of relocation entries within each object module. Text-relocation entries reference unknown values in the text segment of the module. Data-relocation entries reference unknown values in the data segment of the module. Each entry contains the location of the unknown value and the ordinal of the symbol table entry whose address is unknown to the assembler. The loader processes the relocation entries and replaces the unknown values with "hard" addresses.

The text- and data-relocation entries immediately follow the data segment of an object module. If relocation information is present, it amounts to 8 bytes per relocatable datum, as Figure 3-2 indicates.

**Figure 3–2: Format of a Relocation Entry**

```
struct relocation_info {
        int             r_address;      /* address that is relocated */
        unsigned int    r_fill: 4;      /* nothing, yet */
        unsigned int    r_extern: 1;    /* does not include value of sym
                                           referenced */
        unsigned int    r_length: 2,    /* 0=byte, 1=halfword, 2=word */
        unsigned int    r_pcrel: 1;     /* not used */
        unsigned int    r_symbolnum: 24; /* local symbol ordinal */
};
```

where:

*r_address* Offset within the text or data section of a value that requires relocation.

*r_symbolnum* Ordinal symbol table entry number for a symbol whose value is to be used for relocation. The symbol table entries range in ordinal value from zero to a number one less than the number of symbols in the symbol table.

*r_length* Size of the value to be relocated. If the value is 8 bits long, *r_length* = 0; 16 bits long, *r_length* = 1; and 32 bits long, *r_length* = 2.

*r_extern* When the *r_extern* field contains a one, the *r_symbolnum* field is the ordinal of a symbol whose value replaces the unknown value. If the relocation-entry field *r_extern* contains a value of zero, there is no external value required for the unknown value. Instead, the value in *r_symbolnum* is a segment type (e.g., *N_TEXT* for the text segment), and the starting address of that segment relocates the contents of the relocation address.

The *a_trsize* and *a_drsize* fields in the header files give the size (in bytes) of the relocation entries for the text and data segments. If either *a_trsize* or *a_drsize* contains a value of zero, no relocation entries are found in the file for the corresponding text or data segment.

## 3.6 Symbol Table

The symbol table follows the relocation information in the object module. (The location of the symbol table can be computed by the *N_SYMOFF* macro in *a.out.h.*)

In order to resolve the external references between modules discussed above, the assembler creates a list of symbols, or a symbol table, in the object file. The symbol table entries contain all of the information that the assembler knows about the symbol, except its name. The *n_strx* field in the symbol table entry carries the pointer offset to the spelling of a symbol name in the string table. (Refer to section 3.7, "String Table.") The symbol table contains an entry for each symbol name found in the file, whether or not its location can be resolved.

The loader uses the symbol tables in object modules to resolve the references contained in the relocation entries. The *r_symbolnum* field of each relocation entry contains an ordinal number that refers to one of the symbol table entries. The layout of a symbol table entry and the principal flag values that distinguish symbol types are shown in Figure 3-3.

## Figure 3–3: Symbol Table Entry Format

```
struct nlist {
        union {
                char            *n_name;        /* for use when in core */
                long            n_strx;         /* index into file string table */
        } n_un;
        unsigned char           n_type;         /* type flag, i.e N_TEXT;see below */
        char                    n_other;        /* unused */
        short                   n_desc;         /* see <stab.h> */
        unsigned                n_value;        /* value of this symbol */
};
/*
 * Simple values for n_type
 */
#define  N_UNDF  0x0            /* undefined */
#define  N_ABS   0x2            /* absolute */
#define  N_TEXT  0x4            /* text */
#define  N_DATA  0x6            /* data */
#define  N_BSS   0x8            /* bss */
#define  N_COMM  0x12           /* common (internal to ld) */
#define  N_FN    0x1f           /* file name symbol */
#define  N_EXT   01             /* external bit, OR'ed in */
#define  N_TYPE  0x1e           /* mask for all type bits */

/*
 * Other permanent symbol table entries have some of the N_STAB bits
   set. These are given in <stab.h>
 */
```

where:

*n_strx* Contains an offset into the string table, which is a pool of symbol names. This value is relative to the start of the string table.

*n_type* Contains a string designating the type of symbol. (Values for *n_type* are described below.)

*n_other* Unused.

*n_desc* Used for the debugger, *csd*(1).

*n_value* Contains the value of this symbol. Generally, the contents of this field are a resolved address. (Remember that even executable files with all of their references resolved still contain symbol tables.)

Different values for the *n_type* field include the following:

*N_UNDF* Symbol is undefined.

*N_ABS* Symbol is an absolute value that is not associated with any segment.

*N_TEXT* Symbol is in the text segment.

*N_DATA* Symbol is in the initialized data segment.

*N_BSS* Symbol is in the uninitialized data segment.

*N_COMM* Symbol is a FORTRAN COMMON block or C initialized global data symbol. The value of these symbols is the size in bytes of the storage block. This value is used internally in the loader and is never in an executable file.

*N_FN* Symbol is a name whose string table entry is an object file name.

*N_EXT* A bit that is logically ORed with another type to indicate that the symbol is global instead of local.

*N_TYPE* A bit mask that defines all valid symbol table types.

Use *nlist*(3) to examine the symbol table of an object module.

## 3.7 String Table

The *n_strx* field of the symbol table entry contains an offset into the string table. The string table is a pool of null-terminated strings that are symbol name spellings. The string table immediately follows the symbol table in the object module.

# Chapter 4
# Standard Object File Format (SOFF)

## 4.1 Introduction

Versions of the operating system V6.1 and later produce object files in a format called the standard object file format (SOFF). This chapter describes the following features of the SOFF file:

- Header
- Optional header
- Program segment headers
- Segment data
- Relocation information
- Symbol table
- String table

The SOFF format requires that the segments be arranged in the following order: the file header first, then the optional header. All the other segments can be arranged in any order.

| CAUTION | All pointer fields in the SOFF format are specified as *long long* words. Remember to cast them to an *off_t* before using them with routines such as *fseek*(3s) or *lseek*(2). |
|---|---|

## 4.2 File Header

The file header contains general information about the SOFF file, such as the magic number, version number, time and date stamp, number of segments defined, and so forth.

Figure 4-1 shows the layout of the file header.

## Figure 4–1: File Header Layout

```
/*
 *  filehdr.h - Definition for the SOFF file header
 */

#ifndef _filehdr_h                      /* To allow #include multiple times   */
#define _filehdr_h

struct filehdr {
        unsigned long      h_magic;     /* Magic number                       */
        unsigned long      h_version;   /* Object file version number         */
        long               h_timdat;    /* Time stamp of file creation        */
        unsigned long      h_nscns;     /* Number of sections in file         */
        unsigned long long h_scnptr;    /* Ptr to first section header        */
        unsigned long      h_opthdr;    /* Size, in bytes, of optional header */
        unsigned long long h_strptr;    /* Offset in file of string table     */
        unsigned long long h_strsiz;    /* Size, in bytes, of string table    */
        unsigned long long h_flags;     /* Flag word                          */
        long               h_spares[3]; /* Room for growth                    */
};

#define FILEHDR struct filehdr
#define FILEHSZ sizeof(struct filehdr)

/* The magic number in this header defines that this is a SOFF file.
 * The o_flags field in the optional header (see <convex/opthdr.h>) defines
 * what kind of object/executable it is.
 */

/*  Magic number */

#define SOFF_MAGIC 0600
#define USR_SOFF_MAGIC 0601
#define            IS_SOFF_MAGIC(X) ((X) == SOFF_MAGIC)

/* FLAG definitions (h_flags) */

#define H_COMM          0x0000000000000001LL   /* File contains .comm sections */
#define H_RSVDBITS      0x00000fffffffffffLL   /* Reserved bits                */
#define H_USERBITS      0xffffff0000000000LL   /* Reserved for users           */

#endif /* _filehdr_h */
```

where:

*h_magic* SOFF magic number *SOFF_MAGIC* defined in the include file *filehdr.h*. This magic number indicates that this is a SOFF format file. The macro *IS_SOFF_MAGIC* returns a "true" value if the magic number passed to it is a valid SOFF magic number.

*h_version* A version number can be placed in this field by the assembler. It is not a required field, but is set to zero if it is not used. The loader selects the largest version number of all object files processed and uses this value as the version number of the executable file produced. If you specify a version number on the *ld* command line, this number is used regardless of any version numbers found in the object files. You can use the *vers*(1) command to set the version number after the executable file has been produced.

*h_timdat* Current time and date (from *time*(2)) are placed in this field at some time during assembly.

*h_nscns* Number of segment headers written to the object file.

*h_scnptr* Offset, from the beginning of the file, to the start of the first segment header. This value can be used with *fseek*(3s) to locate the segment headers.

*h_opthdr* Size, in bytes, of the optional header in the object file. Currently, this is set to the value *OPTHSZ*, defined in *opthdr.h*.

*h_strptr* Offset, from the beginning of the file, to the start of the string table. This value can be used with *fseek*(3s) to locate the string table.

*h_strsiz* Number of bytes in the string table. It can be used with *fread*(3s) to read the string table.

*h_flags* Table 4-1 shows the flags defined for use in a SOFF file header.

**Table 4-1: File Header Flags**

| Flag | Value | Description |
|---|---|---|
| H_COMM | 0x00000001LL | Set if the file has segment headers that describe initialized common blocks. The assembler sets this flag if it has output a segment of this type. |
| H_RSVDBITS | 0x000000ffffffffffLL | This mask covers all the bits that are reserved for use by CONVEX. |
| H_USERBITS | 0xffffff0000000000LL | This mask covers all the bits that are reserved for users. |

*h_spares* Reserved for future expansion; it is always set to zero.

## 4.3 Optional Header

The optional header is used to store information that is specific to object and executable files. It contains information such as the location of the symbol tables and the entry point for executable files.

> **Note**
>
> The optional header is differentiated from the file header in that the file header stores information specific to the SOFF format only, not its application to object files.

Figure 4-2 shows the layout of the optional header.

## Figure 4–2: Optional Header Layout

```
/*
 * opthdr.h - Definitions for SOFF optional header
 * Be careful when changing this file.  The 68000 C compiler doesn't understand
 * the concept of long long data types.
 */
#ifndef _opthdr_h                              /* to avoid multiple inclusions */
#define _opthdr_h

#ifndef mc68000
  struct opthdr {
        unsigned long long     o_symptr;       /* Offset in file of symbol table */
        unsigned long          o_nsyms;        /* Num of entries in symbol table */
        long                   o_spare;        /* RESERVED - must be 0 */
        unsigned long          o_entry;        /* Entry point */
        unsigned long long     o_flags;        /* defined below */
  };
#else                                          /* This is for 68000 with no long longs */
  struct opthdr {
        unsigned long          o_symptr_hi;    /* Offset of symbol table (hi  32) */
        unsigned long          o_symptr_lo;    /* Offset of symbol table (low 32) */
        unsigned long          o_nsyms;        /* Num of entries in symbol table */
        long                   o_spare;        /* RESERVED - must be 0 */
        unsigned long          o_entry;        /* Entry point */
        unsigned long          o_flags_hi;     /* defined below */
        unsigned long          o_flags_lo;     /* defined below */
  };
#endif
#define OPTHDR struct opthdr
#define OPTHSZ sizeof(struct opthdr)

/*
 * Flag bits defined in o_flags...
 */

#ifndef mc68000
#define OF_DEMAND      0x0000000000000001LL   /* old ZMAGIC */
#define OF_PREPAGED    0x0000000000000002LL   /* old PMAGIC */

  •
  •
  •

#define OF_EXEC        0x8000000000000000LL
#else                                          /* For 68000, must use o_flags_hi */
#define OF_DEMAND      0x0000000000000001     /* old ZMAGIC */
#define OF_PREPAGED    0x0000000000000002     /* old PMAGIC */

  •
  •
  •
#define OF_HI_EXEC     0x80000000
#endif
#endif                                         /* _opthdr_h */
```

where:

*o_symptr* Offset, from the beginning of the file, to the start of the symbol table. This value can be used with *fseek*(3s) to locate the symbol table entries.

*o_nsyms* Number of symbol table entries written to the symbol table. This value can be used with *fread*(3s) to read the symbol table.

*o_spare* Reserved for future expansion; this field is set to zero.

*o_entry* Store the entry point of the executable image. This field is set by the loader. The assembler always sets this field to zero.

*o_flags* Optional header flags are used to indicate a variety of information about the object and executable files contained in the SOFF file. These flags determine items such as execution modes, IEEE usage, whether the file is stripped, and so on.

Table 4-2 shows the flags defined for use in the optional header.

**Table 4-2: Optional Header Flags**

| Flag | Value | Description |
|---|---|---|
| OF_DEMAND | 0x0000000000000001LL | Set if the executable is to be demand paged. |
| OF_PREPAGED | 0x0000000000000002LL | Set if the executable is to be prepaged. |
| OF_NON_SWAP | 0x0000000000000004LL | Set if the executable is not to be swapped and should be locked in memory. |
| OF_POSIX | 0x0000000000000008LL | Set if the executable is to be POSIX conforming. |
| OF_RESERVED_EXEC_BITS | 0x0000000000000ff0LL | Bits reserved for future expansion of the execution mode flags. |
| OF_EXEC_MASK | 0x0000000000000fffLL | This mask is used to set/extract the executable bits combined. |
| OF_INST_C1 | 0x0000000000000000LL | C1 instruction |
| OF_INST_C2 | 0x0000000000010000LL | C200 Series instruction |
| OF_INST_C2MP | 0x0000000000020000LL0 | C200 Series multiprocessor instruction |
| OF_INST_PARALLEL | 0x0000000000040000LL | Parallel instruction |
| OF_INST_INTRINSIC | 0x0000000000080000LL | Intrinsic instruction |
| OF_INST_MASK | 0x00000000000f0000LL | Instruction mask |
| OF_HDW_IGNORE | 0x0000000000100000LL | Ignore hardware features |
| OF_RESERVED0 | 0x0200000000000000LL | Reserved, zero. |
| OF_NOT_VECTOR | 0x0400000000000000LL | Set if the code in this file does not use vector instructions. Not implemented at the present time. |
| OF_FP_MODE_NATIVE | 0x0000000000000000LL | Value if the code in this file uses only native-mode floating point arithmetic. |
| OF_FP_MODE_IEEE | 0x1000000000000000LL | Value if the code in this file uses only IEEE-mode floating point arithmetic. |
| OF_IEEE_MASK | 0x1800000000000000LL | This mask covers bits used to indicate the IEEE-mode of this object/executable. |
| OF_STRIPPED | 0x2000000000000000LL | Set if the file has been stripped using *strip*(1). The assembler never sets this flag. The loader sets this flag when the *-s* command line flag is processed. |
| OF_OBJECT | 0x4000000000000000LL | Set if the file is an object file. The assembler sets this flag. |
| OF_EXEC | 0x8000000000000000LL | Set if the file is an executable image. The assembler never sets this flag; only the loader does. |

## 4.4 Segment Headers

Segment headers are used to supply information about the various segments defined in the SOFF file. There are five predefined or standard segments—text, data, tdata, bss, and tbss—and one segment for each initialized common block.

A segment header is not written to the SOFF file if the segment it controls was not used. In other words, a segment with a size of zero is not written to the output file.

### 4.4.1 Predefined Segments

The SOFF has five predefined segments, although none of these segments has to exist in any particular object file. These segments handle the standard segments that can occur in object files.

Table 4-3 shows the names of the five predefined segments and the type of data included in each.

**Table 4-3: Predefined Segments**

| Segment Name | Type of Data |
|---|---|
| .text | instruction |
| .data | initialized data |
| .tdata | initialized thread data |
| .bss | uninitialized data |
| .tbss | uninitialized thread data |

These names are placed in the string table, although their location in the table is not important. The segments have names in the string table, but they do not have a symbol associated with them. The names *.text*, *.data*, *.tdata*, *.bss*, and *.tbss* are not reserved symbol names; they can be used as freely as any other symbol name.

For more on program segments, refer to the *CONVEX Assembly-Language User's Guide*, section 4.5.1, "Program-Segment Directives." For information on multithreaded parallel programs, refer to the *CONVEX Architecture Reference*, section 5.4, "Multithreaded Execution (Forking/ASAP)." For information on debugging multithreaded parallel programs, refer to the *CONVEX adb (Assembly-Language Debugger) User's Guide*, chapter 5, "Multithreaded Debugging."

### 4.4.2 Initialized Common Blocks

The name of the segment that handles an initialized common block is the name of the common block. The symbol table entry and the segment header share an entry in the string table.

### 4.4.3 Segment Header Layout

Figure 4-3 shows the layout of the segment headers.

## Figure 4–3: Segment Header Layout

```
/*
 * scnhdr.h - Definition for the SOFF section headers
 * Be careful when changing this file.  The 68000 C compiler doesn't understand
 * the concept of long long data types.
 */

#ifndef _scnhdr_h
#define _scnhdr_h

#define s_align s_vaddr                         /* Also used for alignment data */

#ifndef mc68000                                 /* If not being compiled for 68000 */
  struct scnhdr {
        unsigned long long s_strndx;            /* Section name index in string table */
        unsigned long s_vaddr;                  /* Virtual load address of section */
        unsigned long long s_size;              /* Size of section in bytes */
        unsigned long long s_scnptr;            /* Offset in file to raw data */
        unsigned long long s_relptr;            /* Offset in file to relocation data */
        unsigned long s_nrel;                   /* Number of relocation entries */
        unsigned long s_prot;                   /* Protection flags */
        unsigned long long s_flags;             /* Flags word */
  };
#else                                           /* cc68 doesn't understand long long */
  struct scnhdr {
        unsigned long s_strndx_hi;              /* Sec. name string tbl index, hi 32 */
        unsigned long s_strndx_lo;              /* Sec. name string tbl index, lo 32 */
        unsigned long s_vaddr;                  /* Virtual load address of section */

          •
          •
          •
        unsigned long s_flags_lo;               /* Flags word, low  32 bits */
  };
#endif

/*
 * s_vaddr and s_align are synonyms, s_align is used for alignment data in
 * object files, since they can't have virtual addresses, and s_vaddr is used
 * in executable images, since they already have the alignment data figured
 * in to the load address.
 */

#define SCNHDR struct scnhdr
#define SCNHSZ sizeof(struct scnhdr)


   /* FLAG definitions */

   /* Section protection flags */

#define VM_PG_R         0x08                    /* Section memory is readable */
#define VM_PG_W         0x04                    /* Section memory is writable */
#define VM_PG_E         0x02                    /* Section memory is executable */
```

```
   /* Section type flags */

#ifndef mc68000
#define S_TEXT          0x00000001LL            /* .text section */
#define S_DATA          0x00000002LL            /* .data section */
#define S_TDATA         0x00000003LL            /* unshared data section */

  •
  •
  •
#define S_USERBITS      0xffffff0000000000LL    /* Reserved for users */
#else    /* cc68 and friends don't understand long long data types */
#define S_TEXT          0x00000001              /* .text section */
#define S_DATA          0x00000002              /* .data section */
#define S_TDATA         0x00000003              /* unshared data section */

  •
  •
  •
#define S_FMTMASK       0x0000ff00              /* mask to get format flags */
#endif

   /* Protection flags (for s_prot, above) are defined in <convex/pte.h> */

#endif                                          /* _scnhdr_h */
```

---

where:

*s_strndx* — Offset into the string table to the start of the segment name.

*s_align/s_vaddr* — *s_align* and *s_vaddr* are synonyms. This field serves two purposes. For object files, it is the alignment requested for the segment, specified with the *.align* directive to the assembler. For executable files, it is the virtual address where the segment is loaded. All references to this field use the proper name (*#defines* are used to make the two names synonymous) for the value desired. That is, references to get or set the segment alignment use the *s_align* fieldname, whereas references to the virtual load address of an executable image use the *s_vaddr* fieldname.

Default value for the alignment of any segment is 8. The assembler sets this field to 8 if no *.align* directive is encountered for a given segment. Otherwise, the field is set to the value specified in the *.align* directive.

*s_size* — Number of bytes of data that the segment contains. For text segments, this is the size of the assembled code. For data segments, it is the size of the data defined. For bss segments, it is the size of bss to be allocated. Any segment with a size of zero is not written to the object file.

*s_scnptr*

Offset, from the beginning of the file, to the start of the raw data for the segment. This value can be used with *fseek*(3s) to locate the raw data for the segment. There is no restriction on placement of data in the file, except that it comes after the file and optional headers (the file and optional headers are the first things in the file).

Some segments do not have any data associated with them (for example, bss). For these segments, this field is set to zero. As a consequence, the setup for an object file that has bss is a segment header with the *S_BSS* flag set, a name of *.bss*, a size set to the needed bss area, and a *s_scnptr* field set to zero.

*s_relptr*

Offset, from the start of the file, to the start of the relocation entries for the segment. This value can be used with *fseek*(3s) to locate the relocation entries. If the segment has no relocation information, this field is set to zero.

*s_nrel*

Number of relocation entries written to the file for this segment. This value can be used with *fread*(3s) to read the relocation entries from the file. If the segment has no relocation information, this field is set to zero.

*s_prot*

Protection flags for the segment when loaded into memory. The flags used are defined in *<convex/pte.h>*. Table 4-4 describes these flags.

## Table 4-4: Protection Flags

| Flag | Value | Description |
|---|---|---|
| PG_R | 0x80000008 | Marks pages that contain raw data for the segment to be readable by the running process. |
| PG_W | 0x00000004 | Marks pages that contain raw data for the segment to be writable by the running process. |
| PG_E | 0x00000002 | Marks pages that contain raw data for the segment to be executable. |

*s_flags*

Segment header flags, which are defined in Table 4-5.

**Table 4-5: Segment Header Flags**

| Flag | Value | Description |
|---|---|---|
| S_TEXT | 0x00000001LL | Marks the segment as text. |
| S_DATA | 0x00000002LL | Marks the segment as data. |
| S_TDATA | 0x00000003LL | Marks the segment as unshared data. |
| S_BSS | 0x00000004LL | Marks the segment as bss. |
| S_TBSS | 0x00000005LL | Marks segments as unshared bss. |
| S_COMON | 0x00000006LL | Marks the segment as an initialized common block. |
| S_CONTEXT | 0x00000008LL | Marks the context segment. |
| S_TCONTEXT | 0x00000009LL | Marks the thread context segment. |
| S_TYPMASK | 0x000000ffLL | This mask covers all the bits that are reserved for use by CONVEX . |
| S_FMTMASK | 0x0000ff00LL | Mask to get format flags. |
| S_RSVDBITS | 0x000000ffffffffffLL | This mask covers all the bits that are reserved for use by CONVEX. |
| S_USERBITS | 0xffffff0000000000LL | This mask covers all the bits that are reserved for users. |

## 4.5 Segment Data

Raw data for a segment depends on the type of segment. The format of the raw data is unchanged from that produced by the earlier version of the assembler. The difference, however, is where the data gets written in the file. This difference, though, is of no consequence because the segment header field *s_scnptr* specifies the location, so no fixed position is required.

## 4.6 Relocation Information

Certain portions of any given object module can contain references to addresses whose values are not known when the object file is assembled. For example, instructions in the text segment of the object module can refer to addresses in the data segment. These data segment addresses are relocated (biased) by the final memory base address of the data segment. In cases like these, the loader generates the final addresses of all the unknown values. The assembler communicates these unresolved addresses to the loader via relocation entries. See Figure 4-4.

Each entry contains the location of the unknown value and the ordinal of the symbol table entry whose address is unknown to the assembler. The loader processes the relocation entries and replaces the unknown values with "hard" addresses.

If relocation information is present, it amounts to 8 bytes per relocatable datum, as Figure 4-4 indicates.

## Figure 4–4: Format of a Relocation Entry

```
/*
 * reloc.h - Definition of the SOFF relocation entry
*/

#ifndef _reloc_h
#define _reloc_h

/* Check to see if a.out.h has been included; if so, skip struct declaration */

#ifndef N_BADMAG
struct relocation_info {
        int             r_address;      /* Address that is relocated */
        unsigned int    r_fill:4;       /* Nothing yet               */
        unsigned int    r_extern:1;     /* Symbol is external        */
        unsigned int    r_length:2;     /* 0=byte, 1=half, 2=word    */
        unsigned int    r_pcrel:1;      /* Not used                  */
        unsigned int    r_symbolnum:24; /* Local symbol ordinal      */
};
#endif /* N_BADMAG */

#define RELOC_INFO struct relocation_info
#define RELOC_INSZ sizeof(struct relocation_info)


#endif /* _reloc_h */
```

where:

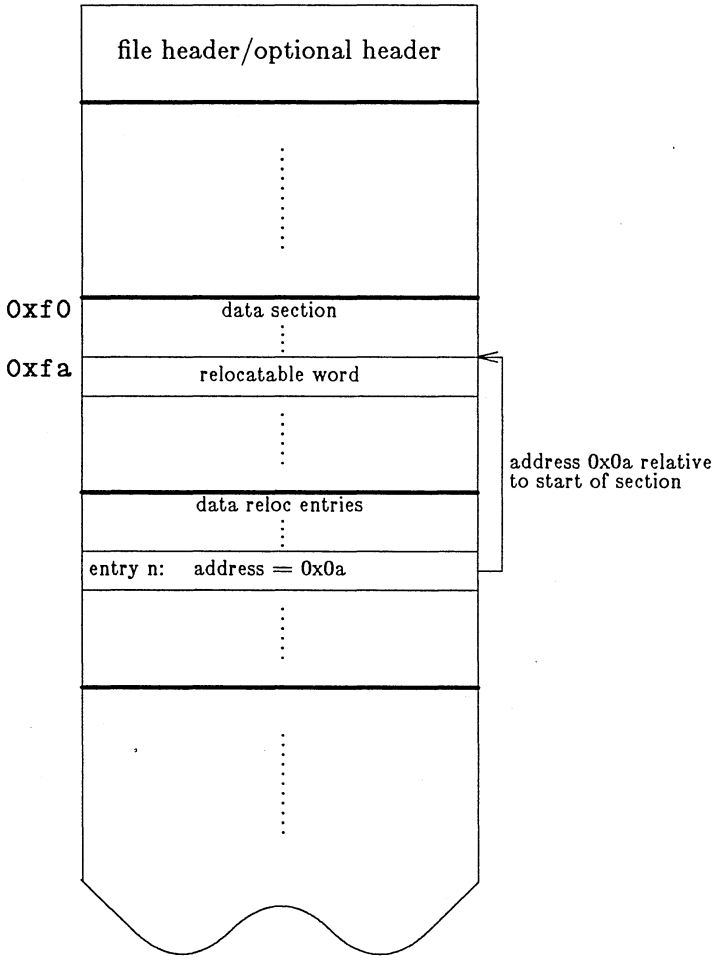
*r_address* — Offset from the beginning of the segment to the byte that starts the reference to the relocatable symbol. Figure 4-5 indicates how this address is used.

*r_symbolnum* — Ordinal symbol table entry number for a symbol whose value is to be used for relocation. The symbol table entries range in ordinal value from zero to a number one less than the number of symbols in the symbol table.

*r_length* — Size of the value to be relocated. If the value is 8 bits long, *r_length* = 0; 16 bits long, *r_length* = 1; and 32 bits long, *r_length* = 2.

*r_extern* — When this field contains a 1, the *r_symbolnum* field is the ordinal of a symbol whose value replaces the unknown value. If the relocation entry field *r_extern* contains a value of zero, no external value is required for the unknown value. Instead, the value in *r_symbolnum* is a segment type (e.g., *N_TEXT* for the text segment), and the starting address of that segment relocates the contents of the relocation address.

**Figure 4–5: Object File**




## 4.7 Symbol Table

The structure used for name list entries is unchanged in the SOFF implementation. There is still one entry for every symbol defined or referenced in any object file. Symbol tables are still used as described in Chapter 3. A flag value for the *n_type* field has been defined. This new flag is used in conjunction with handling common blocks.

Figure 4-6 shows the layout of the symbol table.

## Figure 4–6: Symbol Table Entry Format

```
/*      $CHeader: nlist.h 0.3 88/02/29 20:56:01 $*/
/*      Copyright 1988 Convex Computer Corp.*/

#ifndef __NLIST__
#define __NLIST__       1

/* format of a symbol table entry */

struct nlist {
        union {
                char   *n_name;         /* for use when in core */
                long   n_strx;          /* index into string table */
        } n_un;                         /* union for name and string index */

        unsigned char  n_type;          /* type flag, see below */
        char           n_other;         /* unused */
        short          n_desc;          /* see <stab.h> */
        unsigned long  n_value;         /* value of symbol (or sdb offset) */
};

#define n_hash          n_desc          /* used internally by ld */

/* simple values for n_type */

#define N_UNDF          0x0             /* undefined */
#define N_ABS           0x2             /* absolute */
#define N_TEXT          0x4             /* text */
#define N_DATA          0x6             /* data */
#define N_BSS           0x8             /* bss */
#define N_TDATA         0xa             /* thread data */
#define N_TBSS          0xc             /* thread bss */
#define N_SCNHDR        0x10            /* extended symbol, fully defined */
                                        /* by a section header */
#define N_COMM          0x12            /* un-initialized common */
#define N_FN            0x1f            /* file name symbol */

#define N_EXT           01              /* external bit, or'ed in */
#define N_TYPE          0x1e            /* mask for all the type bits */

/* csd entries have some of the N_STAB bits set. these are given in <stab.h> */

#define N_STAB          0xe0            /* any of these bits set -> csd entry */

/* format for namelist values */

#define N_FORMAT        "%08x"

/* FILE_MOD_FAIL is returned by libc routine nlist() to signify that  */
/* the file being read has been changed in the time it took to do a   */
/* symbol name lookup.                                                */

#define FILE_MOD_FAIL  -2
#endif  __NLIST__
```

where:

*n_strx* Offset into the string table, which is a pool of symbol names. This value is relative to the start of the string table.

*n_type* Value designating the type of symbol (described in Table 4-6).

**Table 4-6: *n_type* Flags**

| Flag | Value | Description |
|---|---|---|
| N_UNDF | 0x0 | The symbol is undefined. |
| N_ABS | 0x2 | The symbol is in the absolute segment. |
| N_TEXT | 0x4 | The symbol is in the text segment. |
| N_DATA | 0x6 | The symbol is in the data segment. |
| N_BSS | 0x8 | The symbol is in the bss segment. |
| N_TDATA | 0xa | The symbol is for thread data. |
| N_TBSS | 0xc | The symbol is for thread bss |
| N_SCNHDR | 0x10 | The symbol is an initialized common block. |
| N_COMM | 0x12 | The symbol is an uninitialized common block. |
| N_FN | 0x1f | File name symbol. |
| N_EXT | 01 | External bit, ORed. |
| N_TYPE | 0x1e | Mask for all the type bits. |

The flag *N_SCNHDR* indicates that the symbol is not fully described by the symbol table entry. It is further described in a segment header.

*n_other* Unused.

*n_desc* Used only by symbol table (STAB) entries, and serves as an index into the segment header table to the header that further defines the symbol. As an example, an initialized common block entered in the symbol table has the *N_SCNHDR* flag set and a segment header created for it, and the *n_desc* field is set to the index of the new segment header. In the segment header, the *S_COMON* flag is set, indicating that it is an initialized common block.

*n_value* Value of this symbol. Generally, content of this field is a resolved address. (Remember that even executable files with all of their references resolved may still contain symbol tables.)

## 4.7.1 *n_value* Field

Using the *n_value* field in the symbol table depends on the type of symbol referenced. The two symbol types are *relocatable* and *common*.

If the symbol is relocatable, the value stored is the offset from the start of the segment in which the word resides. With the old object file format, the value stored was its offset from the beginning of the file; now it is stored as the offset from the start of the segment. This is done to ease the implementation of any utilities that generate SOFF files. Because the symbol values are not relative to their position in the output file, they can be created without first laying out the whole structure of the file. This allows segments to be created in temporary files, then combined into the final output file without having to alter all the symbol values.

Value of a common-block symbol, initialized or uninitialized, is its size in bytes. The assembler keeps track of the largest size specified for any given common block. Remember that an uninitialized common block is indicated by the *n_type* flags *N_UNDF* and *N_EXT* and a nonzero value along with an initialized common block indicated by the flag *N_SCNHDR*; it is further described in a segment header. Refer to section 4.7.2, "Handling Initialized Common Blocks," for more details.

## 4.7.2 Handling Initialized Common Blocks

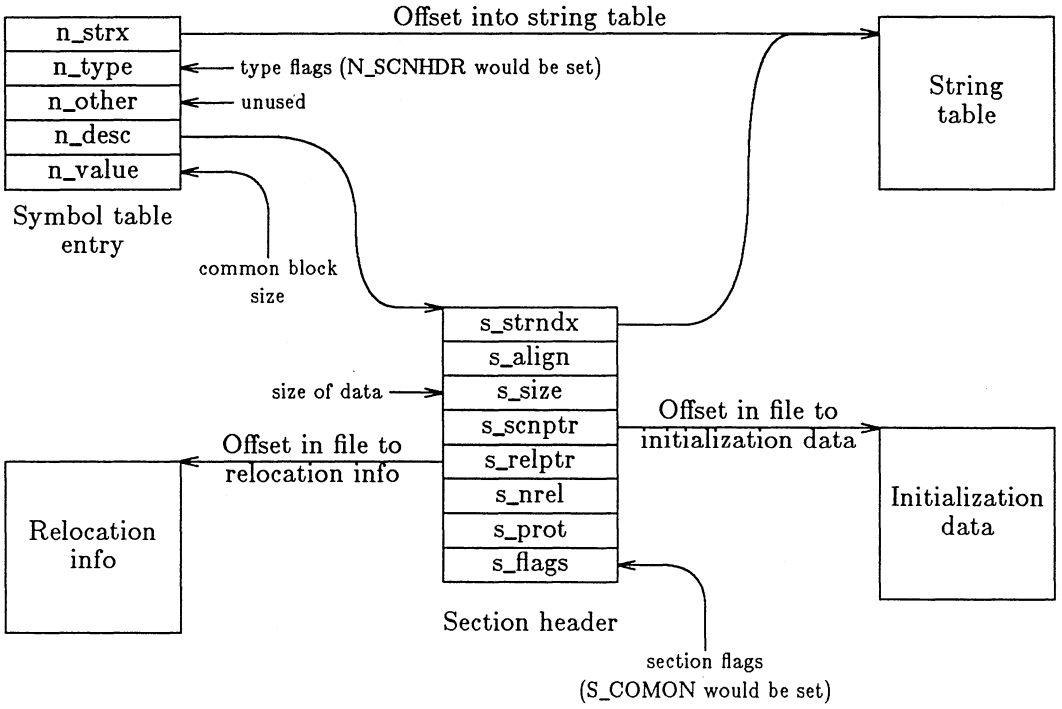
Initialized common blocks (ICBs) are handled differently from ordinary symbols. They are handled more like segments than anything else. There is a symbol in the name list for the common block, and it has an *n_type* flag *N_SCNHDR*. The *n_desc* field is used to store an index into the segment header table for the segment header that describes this common block. This index has its origin at zero and is simply an index into the table of segment headers in the object file.

Each ICB is controlled with a segment header, as though it were a text or data segment. The size of the common block is kept in the *n_value* field of the symbol table entry. The *s_size* field in the segment header indicates the size of the raw data for the ICB. The reason for using two fields to store size information for the ICB is to allow for alternate formats of raw data. If some form of compressed format is used, the size of the raw data may be different from the size of the common block.

The *s_scnptr* field in the header points to the initialization data for the ICB. In this way the initialization data for a given ICB that is initialized in more than one object file can be properly sized and overlaid. ICBs can have relocation entries associated with them. This occurs if any of the initializers are based on relocatable values.

After object files have been loaded into an executable image, the ICBs no longer have segment headers. They have been merged with the data segment, just like any other data symbol. All uninitialized common blocks are rolled into the bss segment. Figure 4-7 illustrates interconnections that exist for an ICB between the symbol table and segment headers.

**Figure 4–7: ICB Symbol Table Interconnections**

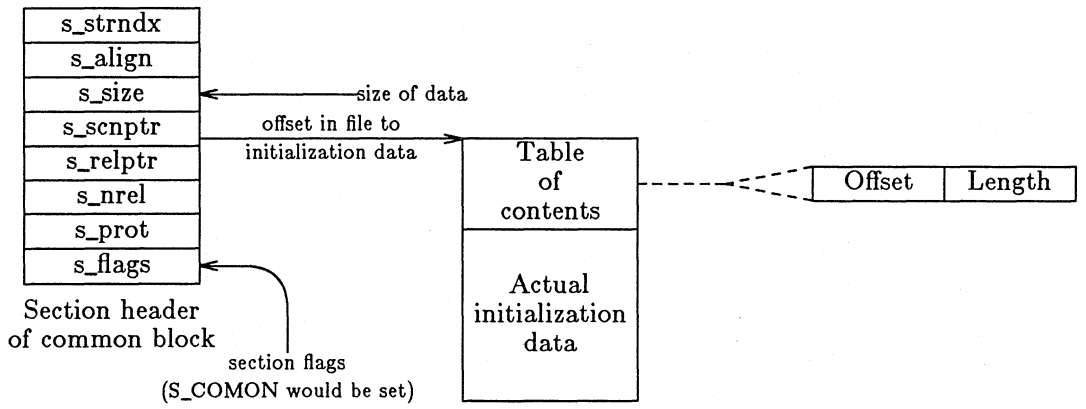



## 4.7.3 Initialization Data Format

To properly keep track of the initializers applied to a particular common block, and to check for multiply-initialized bytes, a "table of contents" of initializers is stored at the front of the raw data segment for an ICB. The first word in the raw data is the number of entries in the table of contents. This table of contents consists of a series of entries containing the offset into the common block and the length of the data initialized (refer to the *scnhdr*(5) man page). The loader uses this data to determine any multiply-initialized bytes, as well as to control the process of overlaying initialization data.

Figure 4-8 illustrates how the raw data is organized.

**Figure 4–8: ICB Raw Data Layout**




## 4.8 String Table

Segment names and symbol names are stored contiguously in the string table as null-terminated character strings. The values stored in *s_strndx* and *n_strx*, for segment names and symbol names, are an offset from the start of the table to the first character in the name. These offsets start at zero for the first name.

In the earlier object file format, the size of the string table was stored as the first word in the string table. This is no longer done. The total size, in bytes, of the string table is now stored in *h_strsiz* in the file header.

# Chapter 5
# Object Libraries

## 5.1 Introduction

This chapter discusses the location and contents of the standard system libraries and the auxiliary programs *ar*, *ranlib*, and *sod*. The *ar* and *ranlib* programs perform archiving and cataloging functions, including the addition and deletion of object modules within libraries. In particular, *ar* creates and updates library files used by the loader, whereas *ranlib* creates the library headers described later in this chapter. The *sod* program enables the user to dump a SOFF file and examine its contents.

## 5.2 Standard System Libraries

System libraries available to the CONVEX user are relocatable libraries. Relocatable libraries are files in which you can store, catalog, and reuse your routines. As the name implies, the final storage locations that the program uses when loaded into main memory are not fixed. Main-storage addresses are fixed when the loader extracts modules.

The contents of the libraries are object modules. A library header contains information enabling the loader to find object modules that contain a definition for an unresolved symbol.

The standard system libraries are in the directories */lib* and */usr/lib* as follows:

| | |
|---|---|
| */lib/libc.a* | C runtime library. This library contains a complete support package for C programming, including system calls, input/output, and math functions. |
| */usr/lib/libm.a* | C math library. |
| */usr/lib/libI77.a* | FORTRAN I/O library. |
| */usr/lib/libF77.a* | FORTRAN intrinsic library; i.e., math and string manipulation routines. |
| */usr/lib/libU77.a* | FORTRAN utility library; i.e., time, date, exit, etc. |
| */usr/lib/libD77.a* | FORTRAN dummy VMS-to-UNIX file name translation routines. |
| */usr/lib/libV77.a* | FORTRAN constants for VAX FORTRAN compatibility. |

Versions of these libraries used with the *prof*(1) and *gprof*(1) profilers have *_p* preceding the library extension (e.g., */usr/lib/libm_p.a*).

## 5.3 Auxiliary Programs

The following sections discuss

- *ar*
- *ranlib*
- *sod*

### 5.3.1 *ar*

The *ar* program is a general-purpose utility that maintains groups of files combined into a single archive file. One use of *ar* is to create and update library files used by the loader. The format of the *ar* command is

**ar** *key* [*posname*] *afile name*

where:

*key* — Refers to the character that enables you to specify which tasks you want *ar* to perform. *key*s differ from switches in that switches must be preceded by a hyphen. *key* is one character from the set *drqtpmx*, optionally concatenated with one or more of *vuaibclo*.

*posname* — Refers to the name of the file used as a marker for positioning the new file in a library.

*afile* — Is the archive file.

*name* — Refers to constituent files within the archive file.

Meanings of *key* characters are:

**c** Creates the named file. Suppresses the message usually produced when *afile* is created. (The message can be very lengthy.)

**d** Deletes the named files from the archive file.

**l** Local. Normally, *ar* places its temporary files in the directory */tmp*. This option places them in the local directory.

**m** Moves the files named to the end of the archive. If a positioning character is present, the *posname* argument must be present.

**p** Prints the named archive files.

**q** Quickly appends the files named to the end of the archive file. Optional key positioning characters are invalid. The command does not check whether the added members are already in the archive. This key helps you avoid excessive processing time when creating a large archive piece by piece.

**r** Replaces the named archive files. If the optional key character *u* is used with *r*, then only those files with "last-modified" dates later than the archive files are replaced. If *a*, *b*, or *i* is used, the *posname* argument must be present. New files are loaded before (use *b* as a key) or after (use *a*) the marker *posname*. If you do not use *posname*, the loader places new files at the end of the library.

**t** Prints a table of contents of the archive file. If you do not specify names, all files in the archive are tabled. If you do specify names, only those files named are tabled.

**v** Verbose. Gives a file-by-file description of the making of a new archive file from the old archive and the constituent file names. When used with *t*, this option gives a long listing of all information about the files. When used with *p*, this option precedes each file with a name.

**x** Extracts the files named. If no names are given, all files in the archive are extracted. In neither case does *x* alter the archive file. Normally, the "last-modified" date of each extracted file is the date when it is extracted. If the *o* switch is used, however, the "last-modified" date is reset to the date recorded in the archive.

There are several reasons why a programmer may want to create a library. Two of the more common reasons are:

- To enhance the loading speed. When the loader links all the modules together, it is much faster for it to process one library than to process many *.o* files from the command line.
- To allow better control of site-specific routines. Several users can use one library file that contains site-specific routines. This avoids lengthy command lines and duplication of the routines among users; it also provides more consistency in the routines used.

This following subsections show how *ar* can be used to create and maintain libraries.

### 5.3.1.1 Creating Libraries

The utility *ar* can be used to create libraries.

**Example 1:**

```
ar q sublib.a m10.o m20.o m30.o m40.o
ar: creating sublib.a
ranlib sublib.a
```

**Example 2:**

```
ar qc sublib.a m10.o m20.o m30.o m40.o
ranlib sublib.a
```

In these examples, you create a new library with the *q* (quick append) key. *ar* creates the library from the *.o* files. These files are placed in the library in the same order that they appear on the command line. The second example uses the *c* key. This instructs *ar* not to print the usual creation message.

### 5.3.1.2 Adding New Modules

When you add a new module to an existing library, you must use the *r* (replace) key. You may also use one of the optional keys *b* (before) or *a* (after). If the *a* or *b* key is used, the *posname* argument must also appear in the command. The new module is inserted after *posname* for *a* or before *posname* for *b*. Following are examples of adding new modules.

**Example 1:**

```
ar rb m40.o sublib.a m35.o
ranlib sublib.a
```

**Example 2:**

```
ar ra m30.o sublib.a m35.o
ranlib sublib.a
```

**Example 3:**

```
ar r sublib.a m45.o
ranlib sublib.a
```

Because the library *sublib.a* was created as previously shown, the first two examples produce identical results in that the first adds new module *m35.o* before module *m40.o* and the second adds new module *m35.o* after module *m30.o*. The third example adds module *m45.o* to the end of the library *sublib.a*.

### 5.3.1.3 Replacing Modules

To replace a module in an existing library, use the *r* (replace) key and simply give the library name followed by the module name to be deleted. If the module does not exist, it is appended to the end of the library file.

**Example:**

```
ar r sublib.a m20.o
ranlib sublib.a
```

### 5.3.1.4 Deleting Modules

To delete a module from an existing library, use the *d* (delete) key and simply give the library name followed by the module name you want to delete. The module names must be the same as they were when the modules were added to the library; that is, the *.o* extension is present.

**Example:**

```
ar d sublib.a m40.o
ranlib sublib.a
```

### 5.3.1.5 Extracting Modules

To extract a module from a library, use the *x* (extract) key. List the modules you want extracted after the library name. In the second example below, no module names are given, so *ar* assumes you want all modules within the given library extracted. This overwrites existing files with no warning. The module names must appear the same as when the modules were put in the library, and it is unnecessary to run *ranlib* after extracting modules from a library because this process does not modify the library.

**Example 1:**

```
ar x sublib.a m10.o
```

**Example 2:**

```
ar x sublib.a
```

### 5.3.1.6 Listing the Names of Modules

To list the names of all modules in a library file, use the *t* (table of contents) key. The next screen example illustrates use of the *t* key and lists sample output from a file named *sublib.a*. Command-line input is in bold type.

```
%ar t sublib.a
m10.o
m20.o
m30.o
m40.o
```

To list the modules and the symbols contained in these modules, use the command *nm*. The next screen example illustrates use of the *nm* command and lists sample output for a file named *sublib.a*.

```
%nm sublib.a
allprint.o:
00000000        a .L1
00000000        a .L2
00000000        a .L3
00000132        t L10000
0000006c        t L13
000001b6        d L15
00000090        t L16
000001b9        d L17
000000b4        t L18
000001bc        d L19
000000d8        t L20
0000015a        t L2000001
000001bf        d L21
000000fc        t L24
000001b0        d L25
00000182        t L30
000001aa        t L9998
00000000        t LL0
                U __flsbuf
00000000        T _allprint
                U _fprintf
00000190        T _printable
00000158        T _sprint
                U _yyout

main.o:
00000000        a .L1
00000000        t LL0
                U _exit
00000000        T _main
                U _yylex
```

•
•
•

## 5.3.2 *ranlib*

*ranlib* converts each archive file to a form that the loader can load more rapidly. It does this by adding a table-of-contents file named *_.SYMDEF* to the beginning of the archive. It uses *ar* to reconstruct the archive. *ranlib* must be run when the library is created and after each time the library file is moved or modified.

The format of the *ranlib* command is

**ranlib** *archivefile* ...

where:

*archivefile* Name of the library file to be loaded.

### 5.3.3 *sod*

*sod* is a utility that dumps the contents of a SOFF file and presents it in human-readable form.

The format of the *sod* command is

**sod** *file* [**-l**] [**-f**] [**-n**] [**-t**] [**-shdr** { *name* or **all** }] ...

where:

*file* Name of the object file you want to examine.

**-l** Turns long-form printing on. Flags in the file header, the optional header, and the section headers are expanded to names that show what is set. The default is to print the flags as hexadecimal values only.

**-f** Prints the file header and the optional header.

**-n** Prints the symbol table (name list).

**-t** Displays a table of contents of file sections. Each entry in the table gives the name of the section, its size, its flag word, the number of relocation entries for the section, and the alignment for the section. If the file has been stripped, the names of the sections are replaced with numeric identifiers that are used to display a particular section with the *-s* commands.

**-s** Displays a file section. If *name* is given, only the named section is displayed. If **all** is used, all file sections are displayed. The modifiers **h**, **d**, and **r** select the format of the display. With no modifiers, *-s* displays only the section header. If **h** is specified, the section header is *not* displayed. If **d** is specified, the section's raw data (program text for the *text* section, for example) is printed. If **r** is specified, the section's relocation information is printed. The *-s* option can be repeated.

> **Note** If the symbol and string tables have been removed from *file* with *strip*(1), *sod* does not display or recognize section names.

If you specify no arguments on the command line, *sod* enters an interactive mode and prompts you for commands. All command-line arguments have the same use and meaning, though *-s* is not recognized.

The following options are available in interactive mode:

**?** Displays the help message.

**c** Prints the full path name of the current working file.

**e** Switches to a new *file*. *sod* displays the last component of the current working file path name in the prompt. When you change to a different file, the prompt also changes to show the new path name.

**h** Displays the help message, which lists valid options and explains what each option does.

**l** Switches the long form on and off.

**q** Quits *sod*.

# Chapter 6
# Examples

In the simplest case, the loader is used to make an executable program from separately compiled object modules. For example, the command sequence

**ld prog.o moda.o modb.o**

causes the loader to combine modules *prog.o*, *moda.o*, and *modb.o*, and to resolve symbolic references between the modules. The executable program file is named *a.out*.

To access the loader from the C compiler, type

**cc main.c sub1.o sub2.o -o main**

This command sequence instructs *cc* to compile the C program *main.c* and to invoke *ld* to combine *main.o* with the object modules *sub1.o* and *sub2.o*. The *-o* option names the resulting executable program *main*.

Create user libraries with a command sequence similar to

**ar cv mylib.a foo.o bar.o mumble.o; ranlib mylib.a**

where the characters *cv* are *keys*, or options. In this case, *c* instructs *ar* to create the library *mylib.a* with constituent files *foo.o*, *bar.o*, and *mumble.o*. The character *v* instructs *ar* to give a file-by-file description of the making of the library from the constituent files. Invoke *ranlib* to make the library header after the archive is created.

Once the library is created, using it is simple. For example, to load the routine *main.o* with the routines housed in the library *mylib.a*, use the command sequence:

**cc main.o mylib.a**

You can also access the loader using the FORTRAN compiler, *fc*. This is the preferred method of loading compiled object modules because it does not require enumerating the libraries. For example,

**fc -o nora1 main.o plot.o datain.o -lapp1**

creates the program file *nora1* out of object files *main.o*, *plot.o*, and *datain.o*. This program also resolves references to a user library (*/usr/lib/libapp1.a*).

# APPENDIX A
# Error Messages

The following error messages are produced by the loader. These messages are by no means a complete listing of all errors and warnings, just some of the more ambiguous ones.

```
%s: vaddr (%#llx) and d_off (%#llx) disagree
```

The loader prints this message when your program has tried to lay out the virtual addresses of the executable image.

```
__.SYMDEF entry has disappeared from '%s'

Unable to read __.SYMDEF archive entry in '%s'
```

These messages indicate that the library is malformed (__.SYMDEF deals with ranlibed libraries). Contact your system administrator to see if re-ranlibing a library will fix it.

```
Unknown FP mode in o_flags (%llx)

ICB '%s' initializers overlap: %s(%#llx,%#x) and %s(%#llx,%#x)

ICB symbol '%s' has disappeared

ICB symbol not N_SCNHDR type (%s %d)
```

where:

FP Floating Point

ICB Initialized Common Block

The first ICB message above indicates that two separate files attempted to initialize the same elements of a common block. Refer to sections 4.7.2, "Handling Initialized Common Blocks" and 4.7.3, "Initialization Data Format."

```
unable to seek to ICB TOC's in '%s'

unable to write ICB TOC count in '%s'

unable to write ICB TOC entry in '%s'
```

where:

TOC Table of contents, which has to do with the way the loader represents common blocks.

If these error messages appear, it could be a problem with file permissions.

```
bad obj_type() return value (%d)

bad filetype() return value (%d)

make_symbol() called with local symbol (%s)

make_symbol() called with STAB symbol (%s)

Invalid symbol type (%s %d) in reloc_one_symbol()
```

These messages indicate internal loader errors because they mention internal routines. Please use the *contact* utility to report these problems.

```
Symbol entry for common block '%s' is gone
```

This message indicates another internal loader error.

```
external symbol '%s' not found for symvec
```

symvec is a data structure internal to the loader. This is another error that should be reported. Please include files you were trying to link.

# APPENDIX B
# Sample Load Map

```
Load Map produced on Tue Jun  9 14:17:20 1987

Command: -ML -X /lib/crt0.o ldanalyze.o ldarch.o ldfiles.o ldflags.o ldicb.o
ldio.o ldmain.o ldmaps.o ldpass1.o ldpass2.o ldreloc.o ldsyms.o ldutil.o -lc
-o newld

Output: newld

E-modes: OF_DEMAND
FP-mode: NATIVE
Entry:   0x80001000

      +---------Section  Sizes-----------+
      text           data            bss
   0x16000         0x5000         0x3000
     90112          20480          12288


                ---------------------------------------
                |      Library Modules Included       |
                ---------------------------------------

  1. /lib/libc.a
        atol.o, calloc.o, ctime.o, fchmod.o, fpmode.o, ftruncate.o, mmap.o
        noIEEE.o, qsort.o, scanf.o, doscan.o, atof.o, siglist.o, strncmp.o
        printf.o, perror.o, system.o, Ovfork.o, _exit.o, execl.o, execv.o
        execve.o, time.o, index.o, tolower.o, open.o, stat.o, umask.o, ungetc.o
        unlink.o, abort.o, signal.o, wait.o, writev.o, filbuf.o, errlst.o
        read.o, sprintf.o, lseek.o, sigvec.o, strcmp.o, strcpy.o, bzero.o
        gettimeofday.o, stasg.o, fprintf.o, stdiolib_cmux.o, doprnt.o, ctype_.o
        gcvt.o, ecvt.o, cvt.o, strlen.o, strout.o, flsbuf.o, exit.o, close.o
        data.o, fstat.o, isatty.o, gtty.o, ioctl.o, malloc.o, getpagesize.o
        sbrk.o, udiv64.o, urem.o, urem64.o, write.o, bcopy.o, cerror.o


                ---------------------------------------
                |        Object Files Included        |
                ---------------------------------------

 #  name                                    text          data           bss
--- ----                                 ----------    ----------    ----------

  1 /lib/crt0.o                             0x98           0x8             0
                                        0x80001000    0x80017000          0
  2 ldanalyze.o                             0x1bc0         0x4b0           0
                                        0x80001098    0x80017008          0
  3 ldarch.o                                0x938          0x260           0
                                        0x80002c58    0x800174b8          0
  4 ldfiles.o                               0x1f20         0x480           0
                                        0x80003590    0x80017718          0
  5 ldflags.o                               0x2058         0x710         0xc8
```

```
                                0x800054b0    0x80017b98    0x8001c000
 6 ldicb.o                           0x3e8         0xb8             0
                                0x80007508    0x800182a8             0
 7 ldio.o                           0x1610        0x5b8             0
                                0x800078f0    0x80018360             0
 8 ldmain.o                          0x380        0x260             0
                                0x80008f00    0x80018918             0
 9 ldmaps.o                         0x19a0        0x850             0
                                0x80009280    0x80018b78             0
10 ldpass1.o                        0x248         0x88             0
                                0x8000ac20    0x800193c8             0
11 ldpass2.o                        0xd00        0x230             0
                                0x8000ae68    0x80019450             0
12 ldreloc.o                        0xe20        0x1d0             0
                                0x8000bb68    0x80019680             0
13 ldsyms.o                        0x17a0        0x3b0             0
                                0x8000c988    0x80019850             0
14 ldutil.o                         0x690        0x1a0          0x68
                                0x8000e128    0x80019c00    0x8001c0c8
15 /lib/libc.a(atol.o)                0x98             0             0
                                0x8000e7b8             0             0
16 /lib/libc.a(calloc.o)              0xe0          0x58             0
                                0x8000e850    0x80019da0             0
17 /lib/libc.a(ctime.o)               0x848         0x210          0x48
                                0x8000e930    0x80019df8    0x8001c130
18 /lib/libc.a(fchmod.o)              0x10             0             0
                                0x8000f178             0             0
19 /lib/libc.a(fpmode.o)               0x8             0             0
                                0x8000f188             0             0
20 /lib/libc.a(ftruncate.o)           0x10             0             0
                                0x8000f190             0             0
21 /lib/libc.a(mmap.o)                0x50             0             0
                                0x8000f1a0             0             0
22 /lib/libc.a(noIEEE.o)              0x40          0x28             0
                                0x8000f1f0    0x8001a008             0
23 /lib/libc.a(qsort.o)               0x5e0            0          0x10
                                0x8000f230             0    0x8001c178
24 /lib/libc.a(scanf.o)               0xc0             0             0
                                0x8000f810             0             0
25 /lib/libc.a(doscan.o)             0xc88          0xd8             0
                                0x8000f8d0    0x8001a030             0
26 /lib/libc.a(atof.o)                0x350          0xc0             0
                                0x80010558    0x8001a108             0
27 /lib/libc.a(siglist.o)                0         0x270             0
                                         0    0x8001a1c8             0
28 /lib/libc.a(strncmp.o)             0x40             0             0
                                0x800108a8             0             0
29 /lib/libc.a(printf.o)              0x40             0             0
                                0x800108e8             0             0
30 /lib/libc.a(perror.o)             0x128          0x18             0
                                0x80010928    0x8001a438             0
31 /lib/libc.a(system.o)              0x1a0          0x10             0
                                0x80010a50    0x8001a450             0
32 /lib/libc.a(Ovfork.o)              0x40             0             0
                                0x80010bf0             0             0
33 /lib/libc.a(_exit.o)               0x10             0             0
                                0x80010c30             0             0
34 /lib/libc.a(execl.o)               0x20             0             0
                                0x80010c40             0             0
35 /lib/libc.a(execv.o)               0x28             0             0
                                0x80010c60             0             0
36 /lib/libc.a(execve.o)              0x10             0             0
                                0x80010c88             0             0
37 /lib/libc.a(time.o)                0x48             0             0
                                0x80010c98             0             0
```

```
38 /lib/libc.a(index.o)                    0x30            0            0
                                        0x80010ce0            0            0
39 /lib/libc.a(tolower.o)                  0x30         0x60            0
                                        0x80010d10   0x8001a460            0
40 /lib/libc.a(open.o)                     0x10            0            0
                                        0x80010d40            0            0
41 /lib/libc.a(stat.o)                     0x10            0            0
                                        0x80010d50            0            0
42 /lib/libc.a(umask.o)                    0x10            0            0
                                        0x80010d60            0            0
43 /lib/libc.a(ungetc.o)                   0x90            0            0
                                        0x80010d70            0            0
44 /lib/libc.a(unlink.o)                   0x10            0            0
                                        0x80010e00            0            0
45 /lib/libc.a(abort.o)                     0x8         0x58            0
                                        0x80010e10   0x8001a4c0            0
46 /lib/libc.a(signal.o)                   0x50            0            0
                                        0x80010e18            0            0
47 /lib/libc.a(wait.o)                     0x20            0            0
                                        0x80010e68            0            0
48 /lib/libc.a(writev.o)                   0x10            0            0
                                        0x80010e88            0            0
49 /lib/libc.a(filbuf.o)                  0x2d8            0            0
                                        0x80010e98            0            0
50 /lib/libc.a(errlst.o)                 0x1198          0x8            0
                                        0x80011170   0x8001a518            0
51 /lib/libc.a(read.o)                     0x10            0            0
                                        0x80012308            0            0
52 /lib/libc.a(sprintf.o)                  0x90            0            0
                                        0x80012318            0            0
53 /lib/libc.a(lseek.o)                    0x10            0            0
                                        0x800123a8            0            0
54 /lib/libc.a(sigvec.o)                   0x10            0            0
                                        0x800123b8            0            0
55 /lib/libc.a(strcmp.o)                   0x30            0            0
                                        0x800123c8            0            0
56 /lib/libc.a(strcpy.o)                   0x20            0            0
                                        0x800123f8            0            0
57 /lib/libc.a(bzero.o)                    0xa0            0            0
                                        0x80012418            0            0
58 /lib/libc.a(gettimeofday.o)             0x40            0            0
                                        0x800124b8            0            0
59 /lib/libc.a(stasg.o)                    0x70            0            0
                                        0x800124f8            0            0
60 /lib/libc.a(fprintf.o)                  0x48            0            0
                                        0x80012568            0            0
61 /lib/libc.a(stdiolib_cmux.o)            0x20            0            0
                                        0x800125b0            0            0
62 /lib/libc.a(doprnt.o)                 0x1e58         0xc0        0x400
                                        0x800125d0   0x8001a520   0x8001c188
63 /lib/libc.a(ctype_.o)                      0         0x88            0
                                              0   0x8001a5e0            0
64 /lib/libc.a(gcvt.o)                    0x2c8         0x30            0
                                        0x80014428   0x8001a668            0
65 /lib/libc.a(ecvt.o)                     0x68         0x30            0
                                        0x800146f0   0x8001a698            0
66 /lib/libc.a(cvt.o)                     0x438            0        0x280
                                        0x80014758            0   0x8001c588
67 /lib/libc.a(strlen.o)                   0x20            0            0
                                        0x80014b90            0            0
68 /lib/libc.a(strout.o)                  0x2d8         0x58            0
                                        0x80014bb0   0x8001a6c8            0
69 /lib/libc.a(flsbuf.o)                  0x8c0            0        0x100
                                        0x80014e88            0   0x8001c808
70 /lib/libc.a(exit.o)                     0x30            0            0
```

```
                                    0x80015748            0            0
 71 /lib/libc.a(close.o)                0x18            0            0
                                    0x80015778            0            0
 72 /lib/libc.a(data.o)                    0       0x1458            0
                                           0    0x8001a720            0
 73 /lib/libc.a(fstat.o)                0x10            0            0
                                    0x80015790            0            0
 74 /lib/libc.a(isatty.o)               0x38            0            0
                                    0x800157a0            0            0
 75 /lib/libc.a(gtty.o)                 0x28            0            0
                                    0x800157d8            0            0
 76 /lib/libc.a(ioctl.o)                0x10            0            0
                                    0x80015800            0            0
 77 /lib/libc.a(malloc.o)              0x578          0x8         0x80
                                    0x80015810    0x8001bb78    0x8001c908
 78 /lib/libc.a(getpagesize.o)           0x8            0            0
                                    0x80015d88            0            0
 79 /lib/libc.a(sbrk.o)                 0x30          0x8            0
                                    0x80015d90    0x8001bb80            0
 80 /lib/libc.a(udiv64.o)               0xb8            0            0
                                    0x80015dc0            0            0
 81 /lib/libc.a(urem.o)                 0x48            0            0
                                    0x80015e78            0            0
 82 /lib/libc.a(urem64.o)               0xa0            0            0
                                    0x80015ec0            0            0
 83 /lib/libc.a(write.o)                0x10            0            0
                                    0x80015f60            0            0
 84 /lib/libc.a(bcopy.o)               0x100            0            0
                                    0x80015f70            0            0
 85 /lib/libc.a(cerror.o)               0x10            0            0
                                    0x80016070            0            0
```

```
-----------------------------------------
|      Global Symbols by Address        |
-----------------------------------------
```

The cross-reference (xref) column characters are described as follows:

| | |
|---|---|
| * | Indicates in which object file the symbol is defined. |
| Numbers | If present, indicate object files where the symbol is referenced. |
| (ld)* | Indicates the symbol is automatically generated by the loader. |

As an example, 8* would indicate the symbol is defined in object file 8, ldmain.o.

```
symbol name              type     address   xref
-----------              ----  ----------   --------------
start                    TEXT  0x80001000   1*
_moncontrol              TEXT  0x80001092   1*
mcount                   TEXT  0x80001094   1*
_analyze                 TEXT  0x80001098   2*, 8
_proc_archive            TEXT  0x80002c58   3*, 10, 4
_fix_archive             TEXT  0x8000349e   3*, 4
_proc_file               TEXT  0x80003590   4*, 10, 5
_obj_type                TEXT  0x80003b0c   4*, 3
_init_file               TEXT  0x80003dd8   4*, 3
_map_ifile               TEXT  0x800042b4   4*
_proc_object             TEXT  0x800050a0   4*, 3
_end_file                TEXT  0x80005364   4*, 3
_proc_flag               TEXT  0x800054b0   5*, 10
_overlay_icb             TEXT  0x80007508   6*, 12
_GET_FHDR                TEXT  0x80007b76   7*, 4
```

```
_GET_OHDR                TEXT  0x80007bc8   7*, 4
_GET_EXEC                TEXT  0x80007c1e   7*, 4
_GET_SHDR                TEXT  0x80007c70   7*, 4
_GET_SYMTAB              TEXT  0x80007cd2   7*, 12, 4
_GET_STRTAB              TEXT  0x80007dc6   7*, 12, 4
_GET_NINIT               TEXT  0x80007ef0   7*, 4
_GET_ISCNDATA            TEXT  0x80007f36   7*, 12
_GET_RELOC               TEXT  0x80007f86   7*, 12
_GET_OSCNDATA            TEXT  0x80007fe6   7*, 12
_PUT_HEADERS             TEXT  0x8000803e   7*, 11
_PUT_STRTAB              TEXT  0x80008544   7*, 11
_PUT_SYMTAB              TEXT  0x8000868e   7*, 11
_PUT_ICBTOCS             TEXT  0x800088d4   7*, 12
_PUT_RELOC               TEXT  0x80008af6   7*, 12
_PUT_OSCNDATA            TEXT  0x80008c54   7*, 12
_PUT_LSYMS               TEXT  0x80008da4   7*, 13
_main                    TEXT  0x80008f00   8*, 1
_ldsig_exit              TEXT  0x800090b4   8*, 14
_setup_map_options       TEXT  0x80009280   9*, 5
_map_oldstyle            TEXT  0x8000940e   9*, 2
_map_standard            TEXT  0x800094fe   9*, 11
_map_locsyms             TEXT  0x80009c38   9*, 13
_pass1                   TEXT  0x8000ac20   10*, 8
_pass2                   TEXT  0x8000ae68   11*, 8
_reloc_files             TEXT  0x8000bb68   12*, 11
_sym_lookup              TEXT  0x8000c9d2   13*, 12, 11, 7, 3, 2
_make_symbol             TEXT  0x8000ca7c   13*, 5, 4
_check_spec_symbols      TEXT  0x8000d08e   13*, 10, 2
_is_spec_symbol          TEXT  0x8000d2d2   13*, 2
_make_spec_symbols       TEXT  0x8000d3e8   13*, 2
_trace_symbol            TEXT  0x8000d76e   13*, 4
_reloc_symbols           TEXT  0x8000d9b6   13*, 11
_reloc_lsyms             TEXT  0x8000daac   13*, 12
_myalloc                 TEXT  0x8000e128   14*, 13, 12, 6, 5, 4, 3
_mycalloc                TEXT  0x8000e19e   14*, 13, 12, 9, 7, 5, 2
_strxcmp                 TEXT  0x8000e220   14*, 5
_strcnt                  TEXT  0x8000e290   14*, 5
_strsav                  TEXT  0x8000e2d8   14*, 5, 4
_strtsav                 TEXT  0x8000e33e   14*, 13, 2
_prep_fname              TEXT  0x8000e4ba   14*, 13, 12, 9, 6, 4, 2
_get_section_ptr         TEXT  0x8000e5fe   14*, 13, 12, 9
_lderror                 TEXT  0x8000e6a0   14*, 13, 12, 11, 10, 9, 7, 6
                                            5, 4, 3, 2
_atol                    TEXT  0x8000e7b8   15*, 4, 3
_calloc                  TEXT  0x8000e850   16*, 14
_cfree                   TEXT  0x8000e914   16*
_ctime                   TEXT  0x8000e930   17*, 9
_localtime               TEXT  0x8000e962   17*
_gmtime                  TEXT  0x8000ec42   17*
_asctime                 TEXT  0x8000ee8c   17*
_fchmod                  TEXT  0x8000f178   18*, 11
_mth$fpmode              TEXT  0x8000f188   19*, 1
_ftruncate               TEXT  0x8000f190   20*, 11
_mmap                    TEXT  0x8000f1a0   21*, 4
_munmap                  TEXT  0x8000f1b0   21*
_msleep                  TEXT  0x8000f1c0   21*
_mwakeup                 TEXT  0x8000f1d0   21*
_msync                   TEXT  0x8000f1e0   21*
_gen$noIEEE              TEXT  0x8000f1f0   22*, 1
_qsort                   TEXT  0x8000f230   23*, 9
_scanf                   TEXT  0x8000f810   24*
_fscanf                  TEXT  0x8000f836   24*
_sscanf                  TEXT  0x8000f85c   24*, 5
__doscan                 TEXT  0x8000f8d0   25*, 24
__innum                  TEXT  0x8000fca8   25*
```

```
__instr                 TEXT  0x80010272   25*
__getccl                TEXT  0x80010490   25*
_atof                   TEXT  0x80010558   26*, 25
_atof$n                 TEXT  0x80010566   26*
_atof$i                 TEXT  0x80010708   26*
_strncmp                TEXT  0x800108a8   28*, 4, 3
_printf                 TEXT  0x800108e8   29*, 13, 12, 9, 4
_perror                 TEXT  0x80010928   30*, 11, 7, 4, 3
_system                 TEXT  0x80010a50   31*, 3
_vfork                  TEXT  0x80010bf0   32*, 31
__exit                  TEXT  0x80010c30   33*, 70, 31
_execl                  TEXT  0x80010c40   34*, 31
_execv                  TEXT  0x80010c60   35*, 34
_execve                 TEXT  0x80010c88   36*, 35
_time                   TEXT  0x80010c98   37*, 11, 9
_index                  TEXT  0x80010ce0   38*, 5
_tolower                TEXT  0x80010d10   39*, 25, 14, 5
_open                   TEXT  0x80010d40   40*, 12, 11, 10, 4
_stat                   TEXT  0x80010d50   41*, 5
_umask                  TEXT  0x80010d60   42*, 11
_ungetc                 TEXT  0x80010d70   43*, 25
_unlink                 TEXT  0x80010e00   44*, 8
_abort                  TEXT  0x80010e10   45*, 22, 1
_signal                 TEXT  0x80010e18   46*, 31, 8
_wait                   TEXT  0x80010e68   47*, 31
_writev                 TEXT  0x80010e88   48*, 30
__filbuf                TEXT  0x80010e98   49*, 25
__rdchk                 TEXT  0x80011078   49*
_sys_errlist            TEXT  0x80011188   50*, 30
_read                   TEXT  0x80012308   51*, 49, 7, 4, 3
_sprintf                TEXT  0x80012318   52*, 14, 13, 12, 11, 10, 9, 7
                                           6, 5, 4, 3, 2
_lseek                  TEXT  0x800123a8   53*, 11, 7, 3
_sigvec                 TEXT  0x800123b8   54*, 46
_strcmp                 TEXT  0x800123c8   55*, 13, 9
_strcpy                 TEXT  0x800123f8   56*, 14, 9
_bzero                  TEXT  0x80012418   57*, 16, 12
_gettimeofday           TEXT  0x800124b8   58*, 37, 17
stasg                   TEXT  0x800124f8   59*, 13, 6, 4
_fprintf                TEXT  0x80012568   60*, 14, 8, 2
__doprnt                TEXT  0x800125b0   61*, 60, 52, 29
__doprnt$n              TEXT  0x800125d0   62*, 61
__doprnt$i              TEXT  0x800134d8   62*, 61
_gcvt                   TEXT  0x80014428   64*, 62
_ecvt                   TEXT  0x800146f0   65*, 64, 62
_fcvt                   TEXT  0x80014722   65*, 62
__cvt                   TEXT  0x80014758   66*, 65
__cvt$n                 TEXT  0x80014766   66*
__cvt$i                 TEXT  0x80014978   66*
_strlen                 TEXT  0x80014b90   67*, 30, 14, 9, 4, 3
__strout                TEXT  0x80014bb0   68*, 62
_fflush                 TEXT  0x80014e88   69*, 49
__xflsbuf               TEXT  0x80014f1a   69*
__wrtchk                TEXT  0x80015022   69*
__findbuf               TEXT  0x8001510e   69*, 49
__bufsync               TEXT  0x80015240   69*
__flsbuf                TEXT  0x8001528c   69*, 68, 52
__cleanup               TEXT  0x8001560c   69*, 70
_fclose                 TEXT  0x80015648   69*
_exit                   TEXT  0x80015748   70*, 8, 1
_close                  TEXT  0x80015778   71*, 69, 12, 11, 8, 4, 3
_fstat                  TEXT  0x80015790   73*, 69, 4
_isatty                 TEXT  0x800157a0   74*, 69
_gtty                   TEXT  0x800157d8   75*, 74
_ioctl                  TEXT  0x80015800   76*, 75
```

```
_malloc                 TEXT  0x80015810   77*, 69, 16, 14
_free                   TEXT  0x80015abe   77*, 69, 16, 13, 12, 9, 4, 3
_realloc                TEXT  0x80015b18   77*
_getpagesize            TEXT  0x80015d88   78*, 77, 11, 5, 4, 2
_sbrk                   TEXT  0x80015d90   79*, 77
udiv64                  TEXT  0x80015dc0   80*, 62
urem                    TEXT  0x80015e78   81*, 64, 62, 13
urem64                  TEXT  0x80015ec0   82*, 62
_write                  TEXT  0x80015f60   83*, 69, 11, 7
_bcopy                  TEXT  0x80015f70   84*, 77, 12, 6
cerror                  TEXT  0x80016070   85*, 84, 83, 82, 81, 80, 79
                                           78, 76, 73, 71, 59, 58, 57, 54
                                           53, 51, 48, 47, 44, 42, 41, 40
                                           36, 35, 34, 33, 32, 22, 21, 20
                                           18, 1
_environ                DATA  0x80017000   1*, 35
_ofilename              DATA  0x80018920   8*, 11, 9, 7, 5
_arg_list               DATA  0x80018924   8*, 12, 10, 9, 5, 4, 2
_crnt_arg               DATA  0x80018928   8*, 10, 5, 4, 3
_last_mmap_addr         DATA  0x8001892c   8*, 4
_clean_load             DATA  0x80018930   8*, 11, 2
_fpmode                 DATA  0x80018934   8*, 11, 9, 5, 2
_fpmodes_seen           DATA  0x8001893c   8*, 4, 2
_symtr_max              DATA  0x80018940   8*, 13, 5, 4
_max_version            DATA  0x80018944   8*, 11, 4
_icbosdata              DATA  0x80018948   8*, 11, 7, 2
_icbcnt                 DATA  0x8001894c   8*, 11, 7, 2
_image_size             DATA  0x80018950   8*, 2
_numosect               DATA  0x80018958   8*, 11
_obj_seen               DATA  0x8001895c   8*, 5, 4, 2
_exec_seen              DATA  0x80018960   8*, 11, 4, 2
_nexe                   DATA  0x80018964   8*, 4, 2
_set_emodes             DATA  0x80018968   8*, 11, 9, 5
_Emodes                 DATA  0x80018970   8*, 5
_usrlib_max             DATA  0x80018994   8*, 5
_stdlibs                DATA  0x80018998   8*, 5
_ring                   DATA  0x800189a4   8*, 5, 2
_ringsize               DATA  0x800189a8   8*, 2
_ringbase               DATA  0x800189bc   8*, 2
_unresolved             DATA  0x800189d0   8*, 13, 10, 3, 2
_numldsyms              DATA  0x800189d4   8*, 13, 11, 10, 9
_ofsymndx               DATA  0x800189d8   8*, 4
_numlsyms               DATA  0x800189dc   8*, 12, 11, 7, 4
_maxsyms                DATA  0x800189e0   8*, 12, 4
_lsymlen                DATA  0x800189e4   8*, 11, 4
_symbkt_head            DATA  0x800189ec   8*, 13, 9, 7, 2
_crnt_symbkt            DATA  0x800189f0   8*, 13
_strbkt_head            DATA  0x800189f4   8*, 14, 7
_crnt_strbkt            DATA  0x800189f8   8*, 14
_out_strx               DATA  0x800189fc   8*, 14, 11
_combkt_head            DATA  0x80018a04   8*, 13, 2
_crnt_combkt            DATA  0x80018a08   8*, 13
_icbbkt_head            DATA  0x80018a0c   8*, 6
_crnt_icbbkt            DATA  0x80018a10   8*, 6
_dflag                  DATA  0x80018a14   8*, 12, 11, 7, 5, 2
_dbgflag                DATA  0x80018a18   8*, 5
_eflag                  DATA  0x80018a1c   8*, 11, 9, 5
_fflag                  DATA  0x80018a20   8*, 11, 5, 2
_hflag                  DATA  0x80018a24   8*, 5
_mflag                  DATA  0x80018a28   8*, 10, 5
_oflag                  DATA  0x80018a2c   8*, 5
_rflag                  DATA  0x80018a30   8*, 13, 12, 11, 9, 7, 5, 2
_sflag                  DATA  0x80018a34   8*, 12, 11, 5
_tflag                  DATA  0x80018a38   8*, 12, 5, 4
_vflag                  DATA  0x80018a3c   8*, 11, 5
```

```
_wflag                  DATA  0x80018a40   8*, 14, 5
_xflag                  DATA  0x80018a44   8*, 12, 5, 4
_Dflag                  DATA  0x80018a48   8*, 5, 2
_Fflag                  DATA  0x80018a4c   8*, 13, 5, 4, 2
_Hflag                  DATA  0x80018a50   8*, 5, 2
_Mflag                  DATA  0x80018a54   8*, 5
_NLflag                 DATA  0x80018a58   8*, 5
_Rflag                  DATA  0x80018a5c   8*, 5
_Tflag                  DATA  0x80018a60   8*, 5, 2
_Xflag                  DATA  0x80018a64   8*, 13, 9, 5, 4
__sctab                 DATA  0x8001a038   25*
_sys_siglist            DATA  0x8001a1c8   27*, 8
_sys_nerr               DATA  0x8001a518   50*
_trap_mth_err           DATA  0x8001a51c   50*
__ctype_                DATA  0x8001a5e0   63*, 62, 39, 25, 5
__iob                   DATA  0x8001a774   72*, 69, 49, 29, 24, 14, 8, 2
__lastbuf               DATA  0x8001bb74   72*, 69, 49
_realloc_srchlen        DATA  0x8001bb78   77*
minbrk                  DATA  0x8001bb80   79*
curbrk                  DATA  0x8001bb84   79*
_usrlibs                UCOMM 0x8001c988   5*, 8, 5
_ofdata                 UCOMM 0x8001ca50   7*, 11, 9, 8, 7
_hashtab                UCOMM 0x8001cac0   8*, 13
_argcnt                 UCOMM 0x8001e25c   5*, 10, 8, 5
_cfdata                 UCOMM 0x8001e260   4*, 13, 12, 9, 8, 7, 4
_argptr                 UCOMM 0x8001e278   5*, 10, 8, 5
_symtrace               UCOMM 0x8001e27c   5*, 13, 8, 5
_err_msg                UCOMM 0x8001e280   2*, 14, 13, 12, 11, 10, 9, 8
                                           7, 6, 5, 4, 3, 2
_mapopts                UCOMM 0x8001e2e4   2*, 13, 11, 9, 8, 2
_osdata                 UCOMM 0x8001e2f0   2*, 13, 11, 9, 8, 7, 2
_argc                   UCOMM 0x8001e560   5*, 9, 8, 5
_argv                   UCOMM 0x8001e564   8*, 9
_errno                  UCOMM 0x8001e568   4*, 85, 58, 32, 30, 26, 4
__sobuf                 UCOMM 0x8001e570   72*
_end                      BSS 0x8001f000   (ld)*, 79
```

```
---------------------------------------
|       Global Symbols by Name        |
---------------------------------------
```

```
symbol name             type     address   xref
-----------             ----  ----------   --------------
_Dflag                  DATA  0x80018a48   8*, 5, 2
_Emodes                 DATA  0x80018970   8*, 5
_Fflag                  DATA  0x80018a4c   8*, 13, 5, 4, 2
_GET_EXEC               TEXT  0x80007c1e   7*, 4
_GET_FHDR               TEXT  0x80007b76   7*, 4
_GET_ISCNDATA           TEXT  0x80007f36   7*, 12
_GET_NINIT              TEXT  0x80007ef0   7*, 4
_GET_OHDR               TEXT  0x80007bc8   7*, 4
_GET_OSCNDATA           TEXT  0x80007fe6   7*, 12
_GET_RELOC              TEXT  0x80007f86   7*, 12
_GET_SHDR               TEXT  0x80007c70   7*, 4
_GET_STRTAB             TEXT  0x80007dc6   7*, 12, 4
_GET_SYMTAB             TEXT  0x80007cd2   7*, 12, 4
_Hflag                  DATA  0x80018a50   8*, 5, 2
_Mflag                  DATA  0x80018a54   8*, 5
_NLflag                 DATA  0x80018a58   8*, 5
_PUT_HEADERS            TEXT  0x8000803e   7*, 11
_PUT_ICBTOCS            TEXT  0x800088d4   7*, 12
_PUT_LSYMS              TEXT  0x80008da4   7*, 13
_PUT_OSCNDATA           TEXT  0x80008c54   7*, 12
_PUT_RELOC              TEXT  0x80008af6   7*, 12
```

```
_PUT_STRTAB                TEXT  0x80008544   7*, 11
_PUT_SYMTAB                TEXT  0x8000868e   7*, 11
_Rflag                     DATA  0x80018a5c   8*, 5
_Tflag                     DATA  0x80018a60   8*, 5, 2
_Xflag                     DATA  0x80018a64   8*, 13, 9, 5, 4
__bufsync                  TEXT  0x80015240   69*
__cleanup                  TEXT  0x8001560c   69*, 70
__ctype_                   DATA  0x8001a5e0   63*, 62, 39, 25, 5
__cvt                      TEXT  0x80014758   66*, 65
__cvt$i                    TEXT  0x80014978   66*
__cvt$n                    TEXT  0x80014766   66*
__doprnt                   TEXT  0x800125b0   61*, 60, 52, 29
__doprnt$i                 TEXT  0x800134d8   62*, 61
__doprnt$n                 TEXT  0x800125d0   62*, 61
__doscan                   TEXT  0x8000f8d0   25*, 24
__exit                     TEXT  0x80010c30   33*, 70, 31
__filbuf                   TEXT  0x80010e98   49*, 25
__findbuf                  TEXT  0x8001510e   69*, 49
__flsbuf                   TEXT  0x8001528c   69*, 68, 52
__getccl                   TEXT  0x80010490   25*
__innum                    TEXT  0x8000fca8   25*
__instr                    TEXT  0x80010272   25*
__iob                      DATA  0x8001a774   72*, 69, 49, 29, 24, 14, 8, 2
__lastbuf                  DATA  0x8001bb74   72*, 69, 49
__rdchk                    TEXT  0x80011078   49*
__sctab                    DATA  0x8001a038   25*
__sobuf                   UCOMM  0x8001e570   72*
__strout                   TEXT  0x80014bb0   68*, 62
__wrtchk                   TEXT  0x80015022   69*
__xflsbuf                  TEXT  0x80014f1a   69*
_abort                     TEXT  0x80010e10   45*, 22, 1
_analyze                   TEXT  0x80001098   2*, 8
_arg_list                  DATA  0x80018924   8*, 12, 10, 9, 5, 4, 2
_argc                     UCOMM  0x8001e560   5*, 9, 8, 5
_argcnt                   UCOMM  0x8001e25c   5*, 10, 8, 5
_argptr                   UCOMM  0x8001e278   5*, 10, 8, 5
_argv                     UCOMM  0x8001e564   8*, 9
_asctime                   TEXT  0x8000ee8c   17*
_atof                      TEXT  0x80010558   26*, 25
_atof$i                    TEXT  0x80010708   26*
_atof$n                    TEXT  0x80010566   26*
_atol                      TEXT  0x8000e7b8   15*, 4, 3
_bcopy                     TEXT  0x80015f70   84*, 77, 12, 6
_bzero                     TEXT  0x80012418   57*, 16, 12
_calloc                    TEXT  0x8000e850   16*, 14
_cfdata                   UCOMM  0x8001e260   4*, 13, 12, 9, 8, 7, 4
_cfree                     TEXT  0x8000e914   16*
_check_spec_symbols        TEXT  0x8000d08e   13*, 10, 2
_clean_load                DATA  0x80018930   8*, 11, 2
_close                     TEXT  0x80015778   71*, 69, 12, 11, 8, 4, 3
_combkt_head               DATA  0x80018a04   8*, 13, 2
_crnt_arg                  DATA  0x80018928   8*, 10, 5, 4, 3
_crnt_combkt               DATA  0x80018a08   8*, 13
_crnt_icbbkt               DATA  0x80018a10   8*, 6
_crnt_strbkt               DATA  0x800189f8   8*, 14
_crnt_symbkt               DATA  0x800189f0   8*, 13
_ctime                     TEXT  0x8000e930   17*, 9
_dbgflag                   DATA  0x80018a18   8*, 5
_dflag                     DATA  0x80018a14   8*, 12, 11, 7, 5, 2
_ecvt                      TEXT  0x800146f0   65*, 64, 62
_eflag                     DATA  0x80018a1c   8*, 11, 9, 5
_end                        BSS  0x8001f000   (ld)*, 79
_end_file                  TEXT  0x80005364   4*, 3
_environ                   DATA  0x80017000   1*, 35
_err_msg                  UCOMM  0x8001e280   2*, 14, 13, 12, 11, 10, 9, 8
```

```
                                                7, 6, 5, 4, 3, 2
_errno                          UCOMM  0x8001e568   4*, 85, 58, 32, 30, 26, 4
_exec_seen                       DATA  0x80018960   8*, 11, 4, 2
_execl                           TEXT  0x80010c40   34*, 31
_execv                           TEXT  0x80010c60   35*, 34
_execve                          TEXT  0x80010c88   36*, 35
_exit                            TEXT  0x80015748   70*, 8, 1
_fchmod                          TEXT  0x8000f178   18*, 11
_fclose                          TEXT  0x80015648   69*
_fcvt                            TEXT  0x80014722   65*, 62
_fflag                           DATA  0x80018a20   8*, 11, 5, 2
_fflush                          TEXT  0x80014e88   69*, 49
_fix_archive                     TEXT  0x8000349e   3*, 4
_fpmode                          DATA  0x80018934   8*, 11, 9, 5, 2
_fpmodes_seen                    DATA  0x8001893c   8*, 4, 2
_fprintf                         TEXT  0x80012568   60*, 14, 8, 2
_free                            TEXT  0x80015abe   77*, 69, 16, 13, 12, 9, 4, 3
_fscanf                          TEXT  0x8000f836   24*
_fstat                           TEXT  0x80015790   73*, 69, 4
_ftruncate                       TEXT  0x8000f190   20*, 11
_gcvt                            TEXT  0x80014428   64*, 62
_gen$noIEEE                      TEXT  0x8000f1f0   22*, 1
_get_section_ptr                 TEXT  0x8000e5fe   14*, 13, 12, 9
_getpagesize                     TEXT  0x80015d88   78*, 77, 11, 5, 4, 2
_gettimeofday                    TEXT  0x800124b8   58*, 37, 17
_gmtime                          TEXT  0x8000ec42   17*
_gtty                            TEXT  0x800157d8   75*, 74
_hashtab                        UCOMM  0x8001cac0   8*, 13
_hflag                           DATA  0x80018a24   8*, 5
_icbbkt_head                     DATA  0x80018a0c   8*, 6
_icbcnt                          DATA  0x8001894c   8*, 11, 7, 2
_icbosdata                       DATA  0x80018948   8*, 11, 7, 2
_image_size                      DATA  0x80018950   8*, 2
_index                           TEXT  0x80010ce0   38*, 5
_init_file                       TEXT  0x80003dd8   4*, 3
_ioctl                           TEXT  0x80015800   76*, 75
_is_spec_symbol                  TEXT  0x8000d2d2   13*, 2
_isatty                          TEXT  0x800157a0   74*, 69
_last_mmap_addr                  DATA  0x8001892c   8*, 4
_lderror                         TEXT  0x8000e6a0   14*, 13, 12, 11, 10, 9, 7, 6
                                                    5, 4, 3, 2
_ldsig_exit                      TEXT  0x800090b4   8*, 14
_localtime                       TEXT  0x8000e962   17*
_lseek                           TEXT  0x800123a8   53*, 11, 7, 3
_lsymlen                         DATA  0x800189e4   8*, 11, 4
_main                            TEXT  0x80008f00   8*, 1
_make_spec_symbols               TEXT  0x8000d3e8   13*, 2
_make_symbol                     TEXT  0x8000ca7c   13*, 5, 4
_malloc                          TEXT  0x80015810   77*, 69, 16, 14
_map_ifile                       TEXT  0x800042b4   4*
_map_locsyms                     TEXT  0x80009c38   9*, 13
_map_oldstyle                    TEXT  0x8000940e   9*, 2
_map_standard                    TEXT  0x800094fe   9*, 11
_mapopts                        UCOMM  0x8001e2e4   2*, 13, 11, 9, 8, 2
_max_version                     DATA  0x80018944   8*, 11, 4
_maxsyms                         DATA  0x800189e0   8*, 12, 4
_mflag                           DATA  0x80018a28   8*, 10, 5
_mmap                            TEXT  0x8000f1a0   21*, 4
_moncontrol                      TEXT  0x80001092   1*
_msleep                          TEXT  0x8000f1c0   21*
_msync                           TEXT  0x8000f1e0   21*
_mth$fpmode                      TEXT  0x8000f188   19*, 1
_munmap                          TEXT  0x8000f1b0   21*
_mwakeup                         TEXT  0x8000f1d0   21*
_myalloc                         TEXT  0x8000e128   14*, 13, 12, 6, 5, 4, 3
```

```
_mycalloc                TEXT  0x8000e19e   14*, 13, 12, 9, 7, 5, 2
_nexe                    DATA  0x80018964   8*, 4, 2
_numldsyms               DATA  0x800189d4   8*, 13, 11, 10, 9
_numlsyms                DATA  0x800189dc   8*, 12, 11, 7, 4
_numosect                DATA  0x80018958   8*, 11
_obj_seen                DATA  0x8001895c   8*, 5, 4, 2
_obj_type                TEXT  0x80003b0c   4*, 3
_ofdata                 UCOMM  0x8001ca50   7*, 11, 9, 8, 7
_ofilename               DATA  0x80018920   8*, 11, 9, 7, 5
_oflag                   DATA  0x80018a2c   8*, 5
_ofsymndx                DATA  0x800189d8   8*, 4
_open                    TEXT  0x80010d40   40*, 12, 11, 10, 4
_osdata                 UCOMM  0x8001e2f0   2*, 13, 11, 9, 8, 7, 2
_out_strx                DATA  0x800189fc   8*, 14, 11
_overlay_icb             TEXT  0x80007508   6*, 12
_pass1                   TEXT  0x8000ac20   10*, 8
_pass2                   TEXT  0x8000ae68   11*, 8
_perror                  TEXT  0x80010928   30*, 11, 7, 4, 3
_prep_fname              TEXT  0x8000e4ba   14*, 13, 12, 9, 6, 4, 2
_printf                  TEXT  0x800108e8   29*, 13, 12, 9, 4
_proc_archive            TEXT  0x80002c58   3*, 10, 4
_proc_file               TEXT  0x80003590   4*, 10, 5
_proc_flag               TEXT  0x800054b0   5*, 10
_proc_object             TEXT  0x800050a0   4*, 3
_qsort                   TEXT  0x8000f230   23*, 9
_read                    TEXT  0x80012308   51*, 49, 7, 4, 3
_realloc                 TEXT  0x80015b18   77*
_realloc_srchlen         DATA  0x8001bb78   77*
_reloc_files             TEXT  0x8000bb68   12*, 11
_reloc_lsyms             TEXT  0x8000daac   13*, 12
_reloc_symbols           TEXT  0x8000d9b6   13*, 11
_rflag                   DATA  0x80018a30   8*, 13, 12, 11, 9, 7, 5, 2
_ring                    DATA  0x800189a4   8*, 5, 2
_ringbase                DATA  0x800189bc   8*, 2
_ringsize                DATA  0x800189a8   8*, 2
_sbrk                    TEXT  0x80015d90   79*, 77
_scanf                   TEXT  0x8000f810   24*
_set_emodes              DATA  0x80018968   8*, 11, 9, 5
_setup_map_options       TEXT  0x80009280   9*, 5
_sflag                   DATA  0x80018a34   8*, 12, 11, 5
_signal                  TEXT  0x80010e18   46*, 31, 8
_sigvec                  TEXT  0x800123b8   54*, 46
_sprintf                 TEXT  0x80012318   52*, 14, 13, 12, 11, 10, 9, 7
                                            6, 5, 4, 3, 2
_sscanf                  TEXT  0x8000f85c   24*, 5
_stat                    TEXT  0x80010d50   41*, 5
_stdlibs                 DATA  0x80018998   8*, 5
_strbkt_head             DATA  0x800189f4   8*, 14, 7
_strcmp                  TEXT  0x800123c8   55*, 13, 9
_strcnt                  TEXT  0x8000e290   14*, 5
_strcpy                  TEXT  0x800123f8   56*, 14, 9
_strlen                  TEXT  0x80014b90   67*, 30, 14, 9, 4, 3
_strncmp                 TEXT  0x800108a8   28*, 4, 3
_strsav                  TEXT  0x8000e2d8   14*, 5, 4
_strtsav                 TEXT  0x8000e33e   14*, 13, 2
_strxcmp                 TEXT  0x8000e220   14*, 5
_sym_lookup              TEXT  0x8000c9d2   13*, 12, 11, 7, 3, 2
_symbkt_head             DATA  0x800189ec   8*, 13, 9, 7, 2
_symtr_max               DATA  0x80018940   8*, 13, 5, 4
_symtrace               UCOMM  0x8001e27c   5*, 13, 8, 5
_sys_errlist             TEXT  0x80011188   50*, 30
_sys_nerr                DATA  0x8001a518   50*
_sys_siglist             DATA  0x8001a1c8   27*, 8
_system                  TEXT  0x80010a50   31*, 3
_tflag                   DATA  0x80018a38   8*, 12, 5, 4
```

```
_time                    TEXT  0x80010c98   37*, 11, 9
_tolower                 TEXT  0x80010d10   39*, 25, 14, 5
_trace_symbol            TEXT  0x8000d76e   13*, 4
_trap_mth_err            DATA  0x8001a51c   50*
_umask                   TEXT  0x80010d60   42*, 11
_ungetc                  TEXT  0x80010d70   43*, 25
_unlink                  TEXT  0x80010e00   44*, 8
_unresolved              DATA  0x800189d0   8*, 13, 10, 3, 2
_usrlib_max              DATA  0x80018994   8*, 5
_usrlibs                UCOMM  0x8001c988   5*, 8, 5
_vflag                   DATA  0x80018a3c   8*, 11, 5
_vfork                   TEXT  0x80010bf0   32*, 31
_wait                    TEXT  0x80010e68   47*, 31
_wflag                   DATA  0x80018a40   8*, 14, 5
_write                   TEXT  0x80015f60   83*, 69, 11, 7
_writev                  TEXT  0x80010e88   48*, 30
_xflag                   DATA  0x80018a44   8*, 12, 5, 4
cerror                   TEXT  0x80016070   85*, 84, 83, 82, 81, 80, 79
                                            78, 76, 73, 71, 59, 58, 57, 54
                                            53, 51, 48, 47, 44, 42, 41, 40
                                            36, 35, 34, 33, 32, 22, 21, 20
                                            18, 1
curbrk                   DATA  0x8001bb84   79*
mcount                   TEXT  0x80001094   1*
minbrk                   DATA  0x8001bb80   79*
start                    TEXT  0x80001000   1*
stasg                    TEXT  0x800124f8   59*, 13, 6, 4
udiv64                   TEXT  0x80015dc0   80*, 62
urem                     TEXT  0x80015e78   81*, 64, 62, 13
urem64                   TEXT  0x80015ec0   82*, 62

                ----------------------------------------
                |           Local symbols for          |
                |             ldanalyze.o              |
                ----------------------------------------

symbol name              type     address
-----------              ----  ----------
_allocate_common         TEXT  0x80002428
_build_outsect           TEXT  0x80001978
_calc_sizes              TEXT  0x8000212e
_check_unresolved        TEXT  0x80001134
_complete_osdata         TEXT  0x80001d2c
_copyr                   DATA  0x8001700c
_layout_image            TEXT  0x800029ac
_proc_icb                TEXT  0x80001e16
_proc_insect             TEXT  0x80001af2
_rcsid                   DATA  0x80017008
_sanity_checks           TEXT  0x800012d0
_show_fpmodes            TEXT  0x80001720
_show_single_fpmode      TEXT  0x800017da
_update_sizes            TEXT  0x8000276a

                ----------------------------------------
                |           Local symbols for          |
                |               ldarch.o               |
                ----------------------------------------

symbol name              type     address
-----------              ----  ----------
_copyr                   DATA  0x800174bc
_get_arhdr               TEXT  0x8000338c
_init_arcm               TEXT  0x8000310c
```

```
_init_symdef              TEXT  0x80002f6c
_rcsid                    DATA  0x800174b8


                 ----------------------------------------
                 |           Local symbols for          |
                 |              ldfiles.o               |
                 ----------------------------------------

symbol name               type     address
-----------               ----  ----------
_copyr                    DATA  0x8001771c
_filetype                 TEXT  0x80003950
_init_scns                TEXT  0x80004404
_rcsid                    DATA  0x80017718


               ----------------------------------------
               |           Local symbols for          |
               |              ldflags.o               |
               ----------------------------------------

symbol name               type     address
-----------               ----  ----------
_copyr                    DATA  0x80017b9c
_find_library             TEXT  0x800073e4
_get_dnum                 TEXT  0x80006fa4
_get_hnum                 TEXT  0x80006dfc
_get_str                  TEXT  0x80007128
_get_vers                 TEXT  0x800071ec
_rcsid                    DATA  0x80017b98


               ----------------------------------------
               |           Local symbols for          |
               |               ldicb.o                |
               ----------------------------------------

symbol name               type     address
-----------               ----  ----------
_copyr                    DATA  0x800182ac
_new_icbilist             TEXT  0x80007806
_overlay_data             TEXT  0x80007768
_process_icbtoc           TEXT  0x80007550
_rcsid                    DATA  0x800182a8


               ----------------------------------------
               |           Local symbols for          |
               |                ldio.o                |
               ----------------------------------------

symbol name               type     address
-----------               ----  ----------
_copyr                    DATA  0x80018364
_get_data                 TEXT  0x800078f0
_get_word                 TEXT  0x80007a54
_rcsid                    DATA  0x80018360


               ----------------------------------------
               |           Local symbols for          |
               |               ldmain.o               |
               ----------------------------------------
```

```
symbol name                type     address
-----------                ----  ----------
_copyr                     DATA  0x8001891c
_finish                    TEXT  0x800091fe
_init                      TEXT  0x80008fa6
_rcsid                     DATA  0x80018918
_sighup_save               DATA  0x80018a6c
_sigint_save               DATA  0x80018a68
_sigquit_save              DATA  0x80018a70
_usage                     TEXT  0x8000907a

                ----------------------------------------
                |           Local symbols for          |
                |               ldmaps.o               |
                ----------------------------------------

symbol name                type     address
-----------                ----  ----------
_copyr                     DATA  0x80018b7c
_get_type                  TEXT  0x8000a4de
_map_globals               TEXT  0x80009970
_map_header                TEXT  0x800095aa
_map_libs                  TEXT  0x8000a97a
_map_objects               TEXT  0x8000a548
_map_sort_globals          TEXT  0x80009ad4
_qcmp_addr                 TEXT  0x80009f5a
_qcmp_name                 TEXT  0x80009f6e
_qlcmp_name                TEXT  0x80009f98
_rcsid                     DATA  0x80018b78
_show_locsyms              TEXT  0x8000a006
_show_object               TEXT  0x8000a734
_show_sym_table            TEXT  0x8000a1c8

                ----------------------------------------
                |           Local symbols for          |
                |              ldpass1.o               |
                ----------------------------------------

symbol name                type     address
-----------                ----  ----------
_copyr                     DATA  0x800193cc
_need_rescan               TEXT  0x8000ae20
_rcsid                     DATA  0x800193c8

                ----------------------------------------
                |           Local symbols for          |
                |              ldpass2.o               |
                ----------------------------------------

symbol name                type     address
-----------                ----  ----------
_copyr                     DATA  0x80019454
_finish_ofile              TEXT  0x8000ba8c
_layout_file               TEXT  0x8000b23c
_rcsid                     DATA  0x80019450
_reprocess_exec            TEXT  0x8000af2a
_setup_ofile               TEXT  0x8000b6fe
```

```
                    ----------------------------------------
                    |          Local symbols for           |
                    |              ldreloc.o               |
                    ----------------------------------------

symbol name                 type     address
-----------                 ----  ----------
_copyr                      DATA  0x80019684
_make_symvec                TEXT  0x8000c1d0
_rcsid                      DATA  0x80019680
_reloc_one_entry            TEXT  0x8000c346
_reloc_one_file             TEXT  0x8000bcf0
_reloc_section              TEXT  0x8000c0c4
_symvec                     DATA  0x80019688


                   ----------------------------------------
                   |           Local symbols for          |
                   |               ldsyms.o               |
                   ----------------------------------------

symbol name                 type     address
-----------                 ----  ----------
_add_ucb_symbol             TEXT  0x8000d6b2
_copyr                      DATA  0x80019854
_hash                       TEXT  0x8000c988
_make_file_symbol           TEXT  0x8000e0a0
_make_one_spec              TEXT  0x8000d516
_new_symbol                 TEXT  0x8000d5f6
_rcsid                      DATA  0x80019850
_reloc_one_symbol           TEXT  0x8000dd12


                   ----------------------------------------
                   |           Local symbols for          |
                   |               ldutil.o               |
                   ----------------------------------------

symbol name                 type     address
-----------                 ----  ----------
_copyr                      DATA  0x80019c04
_rcsid                      DATA  0x80019c00


                   ----------------------------------------
                   |           Local symbols for          |
                   |        /lib/libc.a(errlst.o)         |
                   ----------------------------------------

symbol name                 type     address
-----------                 ----  ----------
_nptr                       TEXT  0x80011170
```

# APPENDIX C
# Reporting Problems

This appendix introduces the CONVEX Technical Assistance Center (TAC) and *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

## C.1 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

## C.2 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke *contact*, it prompts you for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

## C.3 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- full path name of the program or utility in question
- version number of the program or utility in question

### C.3.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

### C.3.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The next screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility.

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt.

If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, but faster.

### C.3.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The next screen illustrates using the *vers* command to find the version number of the loader (*ld*) utility. Enter **vers**, then the path name of the program or utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

## C.4 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

## C.4.1 Using a *.contact* File

When you invoke *contact*, it prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.

2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

## C.4.2 Aborting the Report

To abort a contact report, either enter the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

## C.4.3 Submitting the *dead.report* File

After you abort a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

**contact -r**

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

## C.4.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, press **fg**. Using CTRL-Z and the *fg* (foreground) command lets you toggle back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to toggle back and forth if you are using the Bourne shell (*sh*).

## C.4.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

### C.4.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

| | |
|---|---|
| **~e** | start the text editor (defined in the EDITOR environment variable) |
| **~h** | display a list of available tilde-escape sequences |
| **~p** | print the contact report to the terminal screen |
| **~r** *filename* | read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response. |
| **~~** | insert a single tilde as the first character in the line |

## C.5 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- name and version of the product
- one-line summary of the problem
- detailed description of the problem
- priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation relating to the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts.

Step 1a To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software or documentation question. The next screen illustrates the *contact* command and the system response.

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b If there is a *.contact* file in your home directory, *contact* skips the first prompt. The next screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory.

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2 The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if you use *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.

Step 3 The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4 The *contact* utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5 The *contact* utility prompts for the priority of the problem. The next screen illustrates this prompt and priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6 The *contact* utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7 The *contact* utility prompts for any other pertinent comments. Please include all relevant information.

Step 8 The *contact* utility prompts for suggestions regarding documentation supporting the product. Indicate whether the documentation could be revised to address the problem.

Step 9 The *contact* utility asks for names of files necessary to reproduce the problem. The next screen illustrates the *contact* prompt and sample user response.

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

**Note**

Tilde-escape sequences are not recognized in responses to this prompt. In *contact*, a tilde in this section means your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If files specified are small text files, they are automatically included in the contact report. If the files are too large to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

Step 10 The *contact* utility prompts you to review, edit, submit, or abort the contact report. The next screen illustrates this prompt.

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review — review the text of the contact report. You are then prompted again to select an option.

Edit — edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.

Submit — sends the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell.

Abort — saves the text of the report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell.

# Index

**CONVEX Loader User's Guide**
Document No. 740-002430-204, Fifth Edition

## Reader's Forum

You are invited to submit your comments concerning the clarity and service of this manual. Constructive critical comments are most welcome and will help us continue in our efforts to generate quality customer documentation. (Please list the page number for questions and comments):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

**From:**

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

**FOR ADDITIONAL INFORMATION/DOCUMENTATION:**

| Location | Phone Number |
|---|---|
| Within continental U.S. | 1(800)952-0379 |
| Outside continental U.S. | Contact local CONVEX office |

Direct Mail Orders to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, Texas 75083-3851 USA

(Fold Here First)


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851

(Fold Here Second)

(Tape or Staple)