mipsABI

MIPS® Processor ABI Conformance Guide Version 1.2.1

MIPS ABI Group, Inc.

IMPORTANT NOTE TO USERS

While every effort has been made to ensure the accuracy of all information in this specification, the contributing vendor companies assume no liability to any party for any loss or damage caused by errors or omissions in this specification.

TRADEMARKS

IRIS is a registered trademark of Silicon Graphics, Inc.
IRIX is a trademark of Silicon Graphics, Inc.
MIPS is a registered trademark of MIPS Technologies, Inc.
OSF/Motif and Motif are trademarks of the Open Software Foundation Inc.
POSIX is a registered trademark of Electrical and Electronics Engineers, Inc.
PostScript is a registered trademark of Adobe Systems, Inc.
RISC/os is a trademark of MIPS Technologies, Inc.
Silicon Graphics is a registered trademark of Silicon Graphics, Inc.
UNIX is a registered trademark.
X Window System is a trademark of the X Consortium, Inc.

Revision History:
1.0 - First formally approved version.
1.0.1 - Minor corrections and changes.
1.1 - First major revision to reflect agreed upon new functionality that is part of the ABI. Added X11R5 and OSF/Motif 1.2 as required interfaces. Added comments about the relationship to OCMP. Introduced Asynchronous I/O and tape ioctls as future directions. Introduced _xstat and _lstat as required interfaces. Introduced extended GOT as future direction.
1.1.1 - Mark IRIX 5.1 and IRIS Development Option 3.17 + ABI changes as the reference platform. Add abicc and abild arguments. Minor cleanup. No functionality changes.
1.1.2 - Mark IRIX 5.2 and IRIS Development Option 3.18 as the reference platform. Revise future directions statement to reflect functionality approved for revision 1.2. No functionality changes.
1.2 - Second major revision to reflect new functionality. Make Asynchronous I/O and the tape ioctl's a standard part of the ABI, moving them from the futures section into Chapter 6. Add new sysinfo call to return the number of processors actually available for assignment. Add comments from a new gABI on the order of invocation of .init/.fini sections. Make the expanded GOT part of the ABI, moving it from the futures section. Add the ELF relocation flags R_MIPS_LOVENDOR and R_MIPS_HIVENDOR to allow for more orderly addition of relocation types. Add sbrk to libc. Add RTLD_GLOBAL option to dlopen() routine. Add libgen as a supported archive library. Add a number of new future direction indications.
1.2.1 - Minor changes. Mark IRIX 5.3 and IRIS Development Option 3.19 as the new reference platform for 1.1; no RP yet for 1.2. Clarify ABI_MTWEOF behavior in mtio section. Deprecate -q option to abicc. Define O_DSYNC and O_RSYNC in async I/O section.

Contents

Figures and Tables

1 INTRODUCTION

Overview

The information contained in this specification provides implementation guidelines and defines additional capabilities beyond the binary standards provided by the UNIX System V Application Binary Interface (the generic ABI or gABI) and the MIPS Processor Specific ABI (psABI), which are required for the development and distribution of shrinkwrap software.

The vendors contributing to the development of this Conformance Guide include: Concurrent Computers, Control Data Systems Incorporated, Dansk Data Elektronik A/S, Ing. C. Olivetti & Co. SpA, MIPS Technologies Incorporated, NCR, NEC Corporation, Pyramid Technology Corporation, Silicon Graphics Incorporated, Siemens Nixdorf Informationssysteme AG, Sony Microsystems, Tandem Computers, Unisoft Group Limited, and UNIX International. These vendors are known collectively as the MIPS ABI group. UNIX System Laboratories (now UNIX Systems Group, Novell, Inc.) has supported the effort through review and comment on the specifications as they were developed.

ISVs contributing to the development of this Conformance Guide include Oracle Corporation and SAS Institute Incorporated.

Audience and Purpose

This specification is intended to guide application writers in developing software that will run portably across all MIPS® RISC processor-based systems that are ABI compliant. It also is a specification for implementors of compliant operating systems. This document is intended to define areas beyond the current ABIs as well as address any outstanding ambiguities. Some information may not be relevant to all audiences.

It is intended that application writers and ISVs use the gABI and the psABI in conjunction with this specification as a complete and rich environment for portable application development on the MIPS ABI reference processor.

Relationship to OCMP

Several of the member firms of the MIPS ABI group are also active in the Open Computing environment for MIPS Platforms (OCMP) efforts in Japan. The OCMP-ABI is based on the MIPS ABI version 1.0.1 with extensions in several areas:

- Internationalization and localization, especially in Japanese language support
- CD-ROM access libraries
- Specification of X11R5 and OSF/Motif 1.2.

The intent of both the MIPS ABI group and the OCMP members is to keep their respective specifications aligned. Version 1.1 of this document introduced X11R5 and OSF/Motif 1.2 as required interfaces. The X11R5 interface is supported by Dynamic Shared Object libraries, while OSF/Motif 1.2 is supported by a static archive, `libXm.a`, present in the ABI build environment. Internationalization and CD-ROM access libraries are possible future additions to the MIPS ABI, but are not yet formally part of it.

Organization

This specification is organized similarly to other ABIs for ease of use with the companion standards. In addition to the standard ABI, this specification covers conformance testing issues, implementation guidelines, and future directions.

This book is organized as follows:

- Chapter 1 provides overview information, including a summary of additions to the ABI and an explanation of conformance issues.
- Chapter 2 describes software installation and packaging issues.
- Chapter 3 presents low level system information.
- Chapter 4 describes object files.
- Chapter 5 describes program loading and dynamic linking.
- Chapter 6 presents library information.
- Chapter 7 describes formats and protocols.
- Chapter 8 describes system commands.
- Chapter 9 describes execution environments.
- Chapter 10 describes windowing and terminal interfaces.
- Chapter 11 discusses development environments for an ABI system.
- Appendix A discusses future directions.

Related Documentation

The following books should be used for reference.

- *System V Application Binary Interface, Revised First Edition, Prentice-Hall, ISBN 0-13-880410-9*. This is the defining document for the generic ABI.
- *System V Application Binary Interface, MIPS Processor Supplement, Prentice-Hall, ISBN 0-13-880170-3*. This is the defining document for the psABI for the MIPS Processor family.
- *System V Interface Definition Third Edition, UNIX System Laboratories, UNIX Press* The SVID specifies the base operating system environment for the development of compatible applications, including the API (Application Programming Interface). It was originally published in four volumes, two additional volumes were published later. UNIX Press Titles are available from Addison-Wesley. The current issues of these volumes are:
    - *Volume I, ISBN 0-201-56652-4*
    - *Volume II, ISBN 0-201-56653-2*
    - *Volume III, ISBN 0-201-56654-0*
    - *Volume IV, ISBN 0-201-56655-9*

- □ *Volume V, ISBN 0-201-56656-7*
- □ *Volume VI, ISBN 0-201-52480-6*
- □ *Set, Volumes I-VI, ISBN 0-201-58822-6*

The *Errata* for the SVID, Third Edition should also be obtained.

- *System V Interface Definition, Fourth Edition*. The 4th edition of the SVID will be published in electronic form by Novell, Inc. It will be available through the Novell "home page" on the World Wide Web, using the URL `http://www.novell.com/`. The 4th edition is not specifically a base document for this specification, but should be consulted to fill in definitions of certain routines defined in this specification especially because they were omitted from the SVID 3rd edition. The places where this is applicable are noted in the text.
- *MIPS RISC Architecture, Kane & Heinrich, Prentice Hall, ISBN 0-13-590472-2*.
- *X Window System: the Complete Reference to Xlib, X Protocol, ICCCM, XLFD, X Version 11 Release 5 Third Edition, Scheifler et al, Digital Press, ISBN 1-55558-088-2*.
- *OSF/Motif Programmer's Reference, Revision 1.2, Open Software Foundation, Prentice-Hall, ISBN 0-13-643115-1*.
- *Portable Operating System Interface (POSIX) IEEE Std. 1003.1b-1993, IEEE, ISBN 1-55937-375-X*. This volume is more formally titled *IEEE Standard for Information Technology – Portable Operating System Interface (POSIX®) – Part 1: Application Program Interface (API) – Amendment 1: Realtime Extension [C Language]*. It provides the base IEEE Std 1003.1-1990 specification, as well as the real-time extensions that were the work of the P1003.4 committee.

Single copies of the Prentice Hall books may be obtained from Simon and Schuster by calling 800-947-7700 (US only) or through the Prentice Hall WWW page as `http://www.prenhall.com/`.

Terminology

The following terms are used throughout this book.

ABI	A specification conforming to the *System V Application Binary Interface*. The *System V ABI* for a particular system is composed of the generic ABI, the processor-specific supplement for the processor used in the system, and the processor-specific conformance guide (this volume) used in the system.
ABI-conforming program	A program written to include only the system routines, commands, and other resources included in the ABI; a program compiled into an executable file that has the formats and characteristics specified for such files in the ABI; a program whose behavior complies with the rules given in the ABI.
ABI-conforming system	A computer system that provides the binary system interface for application programs described in the *System V ABI*.
ABI-nonconforming program	A program written to include system routines, commands, and other resources not included in the ABI; a program compiled into a format different from that specified in the ABI; a program whose behavior does not comply with the rules given in the ABI.

Generic ABI — The processor-independent portions of the *System V Application Binary Interface*. Also abbreviated to "gABI".

Processor-specific ABI or Supplement — Those portions of the *System V ABI* specific to a particular processor architecture. Also abbreviated to "psABI".

Conformance Guide — (this document) Implementation guidelines and additional capabilities beyond the gABI and psABI. Also informally known as the "Black Book".

Reference Platform — The combination of hardware and software such that ABI-conforming applications which are generated on and execute correctly on the reference platform are guaranteed to execute correctly on all ABI-conforming systems. The reference platform hardware for the MIPS ABI group consists of the Silicon Graphics family of workstations and servers, including:

- Personal IRIS Workstations
- INDY Workstation
- INDIGO and INDIGO R4000 Workstations
- CRIMSON Systems
- Power Series Servers
- CHALLENGE Servers
- ONYX Workstations

The reference platform software corresponding to the consists of a reference operating system (IRIX 5.3) and compiler suite (IRIS Development Option 3.19), plus a special MIPS ABI patch set. A future revision of this software will be the Reference Platform for Version 1.2.

Note that this version of the ABI is based on the R3000 instruction set, although compatible applications may be built and executed on all supported MIPS processors.

Undefined behavior — Behavior that may vary from instance to instance or may change at some time in the future. Some undesirable programming practices are marked in the ABI as yielding undefined behavior.

Unspecified property — A property of an entity that is not explicitly included or referenced in the ABI and may change at some time in the future. In general, it is not good practice to make a program depend on an unspecified property.

Conformance

The collection of specifications referenced in this volume, including the gABI, the psABI, and this specification, as well as the X Window System and OSF/Motif volumes, define the standard interfaces for compiled application programs on the MIPS architecture. |

NOTE

Diffmarkings have been inserted in this book to indicate the evolution of the Conformance Guide.

A "|" character in the right-hand margin indicates a change in the ABI made when Version 1.2 of the Conformance Guide was published. Deletions are marked with a "*" character. Note that there were considerable editorial changes (to fonts, layout, table headings, etc.) between these two revisions, the change bars are intended to highlight substantive changes only. Some entirely new sections are not diffmarked, and are so noted. Tables of symbols are also not diffmarked, instead a "*" is used to indicate new symbols.

Differences between Version 1.0 and Version 1.1 of the Conformance Guide are not noted.

General Guidelines for Porting to MIPS ABI Platforms

A document entitled *Constructing MIPS ABI Compliant Binaries on the IRIX 5.x Operating System* is available. This porting guide includes information on how to port applications to the reference platform, as well as a summary of common problems encountered when porting applications. This guide is available from the Silicon Graphics Developer Program or any of the other companies participating in the MIPS ABI group. An informal list of "Frequently Asked Questions" is also available. |

Working with the MIPS ABI Group

A reference paper describing the MIPS ABI group is available. It includes general information about the group as well as a list of marketing and technical representatives from each company. Application developers who wish to develop MIPS ABI compliant applications should contact the Applications manager at any one of the MIPS ABI companies to get detailed information on developer and marketing programs.

Platform Compliance Testing

All MIPS ABI conforming systems are measured for compliance against the MIPS ABI PVTS (Platform Verification Test Suite). This test suite is maintained and enhanced by Silicon Graphics under the supervision of the MIPS ABI Technical Committee. (Significant prior work on PVTS was done by Sony Microsystems, and by UniSoft Group Ltd.) Compliance testing ensures that the interfaces provided by the vendors manifest the specified semantics and syntax. The PVTS also verifies features contained in the ABI documents that may not have programmatic interfaces but may still be needed to satisfy binary compatibility. The group also performs multiple platform testing of selected applications to ensure that "real world" compatibility is achieved. |

The Application Compliance Testing Tool (ACT)

The reference platform operating system provides an Application Compliance Testing tool to check the ABI compliance of a binary program. The tool is provided as an aid in identifying problem areas in producing a compliant binary; passing the tests provided by the tool does not ensure complete conformance.

`check_abi_compliance` — Determines whether a binary references anything which is not defined by the ABI as well as making certain that it supplies variables which are required by the ABI.

`check_abi_interface` — Determines if a shared object supplies all public routines which are required of it by the ABI. It will also list public routines which the shared object supplies which are not part of the ABI.

`check_for_syscalls` — Finds any system calls in a binary (i.e., invocations of the procedure `syscall`). Binaries which contain system calls are inherently not ABI compliant.

2 SOFTWARE INSTALLATION AND PACKAGING

Installation

The requirements in the generic ABI (see ''File Tree for Add-on Software'' in the ''Software Installation'' chapter of the generic ABI) should be expanded to include the notion that an optional product should be structured into the following hierarchy:

`/opt/`*pkg*`/bin`	executables invoked by users
`/opt/`*pkg*`/lib`	libraries and other executables
`/opt/`*pkg*`/man`	manual pages
`/opt/`*pkg*`/include`	include files

where *pkg* is the abbreviated name of an add-on software package. Additionally, a MIPS ABI platform will provide the following directories:

```
/opt/bin
/opt/lib
/opt/man
/opt/include
```

The installation script of an installable package may symbolically link files from directories under `/opt/pkg/*` to the corresponding directory under `/opt`. For example, the post installation script for a package called `xtank`, might contain the following instructions:

```
for file in /opt/xtank/bin/*
do
     ln -s $file /opt/bin
done
```

This is especially important for manual pages in order to limit the proliferation of directories containing manual pages.

NOTE ISVs must be aware of the potential name space conflicts in the `/opt` directory tree. Not only must package *names* be unique, but individual binaries must also have names likely to be unique, since each package may link its files into `/opt/bin`, `/opt/lib`, etc.

Devices

The following devices may be available:

`/dev/abi/floppy`	floppy disk (1.44mb)
`/dev/abi/qic`	QIC-120 cartridge tape
`/dev/abi/qic.nr`	QIC-120 cartridge tape (non-rewinding version)
`/dev/abi/9track`	6250 bpi 9-track tapes
`/dev/abi/9track.nr`	6250 bpi 9-track tapes (non-rewinding version)
`/dev/abi/cd_iso`	CD-ROM in ISO 9660 format

NOTE The devices listed here are supported for installation purposes only and are not guaranteed to generate installation media. Also, these `/dev/abi` nodes will exist only if the device is supported on the system.

3 LOW-LEVEL SYSTEM INFORMATION

Low-Level System Information

NOTE

There are no extensions to low-level system information in this volume.

4 OBJECT FILES

Relocation

NOTE This section is new as of Version 1.2 of the Conformance Guide, but will not be marked with diffmarks.

The MIPS processor specific ABI specifies a Global Offset Table (GOT) which limits a DSO to 16K external references. This has proven to be a limitation for a number of developers who wish to create very large DSOs. A future version of the processor specific ABI expands the GOT. This expanded GOT is sufficiently desirable to incorporate it into Version 1.2 of this document.

The expanded GOT is defined by four new Relocation Types which are added to `<elf.h>`.

Figure 4-1: Expanded GOT Relocation Types

Name	Value	Field	Symbol	Calculation
`R_MIPS_GOTHI16`	22	T-hi16	external	(G - (short)G) >> 16 + A
`R_MIPS_GOTLO16`	23	T-lo16	external	G & 0xffff
`R_MIPS_CALLHI16`	30	T-hi16	external	(G - (short)G) >> 16 + A
`R_MIPS_CALLLO16`	31	T-lo16	external	G & 0xffff

The following paragraph replaces the third from last paragraph of Chapter 4 of the MIPS processor specific ABI.

> The first instance of `R_MIPS_GOT16` or `R_MIPS_GOTHI16` and `R_MIPS_GOTLO16`, or `R_MIPS_CALL16` or `R_MIPS_CALLHI16` and `R_MIPS_CALLLO16` relocations cause the link editor to build a global offset table if one has not already been built.

The following paragraph replaces the last paragraph of Chapter 4 of the processor specific ABI.

> `R_MIPS_CALL16` or `R_MIPS_CALLHI16` and `R_MIPS_CALLLO16` relocation entries load function addresses from the global offset table and indicate that the dynamic linker can perform lazy binding. See ''Global Offset Table'' in Chapter 5 of the processor specific ABI.

To allow vendors flexibility in defining new relocation types for their own internal purposes, the following two definitions are added to `<elf.h>`:

```
#define R_MIPS_LOVENDOR         100     /* Vendor specific relocations */
#define R_MIPS_HIVENDOR         127
```

Vendors are free to use any value between 100 and 127 for such internal purposes. Values in this ranges will not be used for ABI purposes in the future.

Link Editing

ISVs are cautioned that products that compile and link edit themselves after installation, or are designed to be on-site linked with end-user developed code, may inadvertently use system interfaces that are not defined by the ABI. Specifically, the headers and static libraries on the Reference Platform, not specified by the ABI, may not be compatible with those headers and libraries on all ABI vendors' platforms.

This problem occurs only when an ISV's application is link edited after installation (i.e., uses the `ld` command to build itself). If all executables and shared libraries are built on the Reference Platform and pass the Application Compliance Testing tool (ACT), the only possible external interfaces are those specified by the ABI. (Note that ACT does not test manifest constants, such as `ioctl` calls compiled into the program.) The ABI interfaces are present on all ABI conforming Operating Systems in the form of dynamic shared libraries. Thus they will be resolved through dynamic linking to the dynamic shared libraries at run-time.

In the case where an application link edits itself after installation, it may expose an incompatibility between non-ABI headers or libraries on the Reference Platform and another vendor's platform. Furthermore, vendors may change the interface or implementation of libraries not covered by the ABI in new operating system releases.

The MIPS ABI Group offers the following guidelines to ISV's whose applications must link edit after installation; they are not meant to be either exhaustive or foolproof:

- Compiler and linker flags are known to differ across the various platforms. To aid the ISV, each vendor has agreed to provide a consistent interface in the wrappers `abicc` and `abild`, which are defined further in ''Compiler and Linker Options'' in this chapter.
- The math routines in `libm.a` will be consistent across all vendor's platforms.
- `libcurses.a` is known to depend on private interfaces in `libc.so.1` and should be avoided if possible, but limited functionality may be used from it if care is taken.
- Implementations of `libucb.a` may not be compatible across all vendor's platforms and should not be used.
- ISV's should document the version of any system interface used by their application that is not supported by the ABI. For example, applications using X widgets, X extensions, Motif or other system libraries which must be linked statically should document the release level of these libraries. This will allow customers to compare release levels between the application and their system.
- Not all compilation systems behave the same with respect to conflicts with the ANSI C Strict Ref/Def Initialization model (see section 3.1 of the ANSI C Standard and elaboration in 3.1.2.2 of the Rationale for ANSI C; and note that section numbering is different in the ISO version). Some platforms emit warnings from the loader, while others a fatal error. If code is generated on a platform of the former type, it may link on that platform but not on some others. Application source code should be modified such that each external data symbol is defined in exactly one place, done by omitting the `extern` keyword. All other references should use the `extern` keyword.

Object Files

All `.o` files shipped with an ISV product must use position independent code, as described in chapter 3 of the *System V ABI, MIPS Processor Supplement*.

Debugging

Only an ELF symbol table is needed for assembly level debugging with symbols.

Compiler and Linker Options

The various platforms use different flags and switches for the compiler and linker. This is a reflection of the differences in the heritage of the compiler systems on the various machines. Platform vendors should support either the standard SVR4 Software Generation System (SGS) flags and switches or provide a wrapper function which translates these standard switches into the local equivalents. The wrappers `abicc` and `abild` have been developed to permit uniform compilation and loader flags. Vendors should provide these wrappers or equivalent functions. Some compilation systems may not support all the flags and switches that the SVR4 SGS does. Where possible the unsupported switches should be silently ignored by the wrapper or the base compilation system.

The `abicc` wrapper supports the following flags, many are described in more detail in the `cc(SD_CMD)` page in the *System V Interface Definition, Third Edition (SVID 3)*.

- –A controls #`assert` mechanisms.
- –B symbolic, –Bsymbolic, –B dynamic, –Bdynamic, –B static, –Bstatic controls linking phase library name lookup.
- –C controls comment removal.
- –c suppresses linking phase.
- –D controls #`define` mechanisms.
- –dn, –d n, –dy, –d y selects dynamic (*y*) or static (*n*) linking.
- –E preprocess only (no compilation or linking).
- –f obsolete, ignored.
- –G direct linker to produce shared object.
- –g generate symbolic debug information.
- –H prints include file names as they are processed.
- –I controls include file lookup.
- –Jsfm not supported, silently ignored.
- –Kfpe, –K fpe, –Kmau, –K mau, –Ksd, –K sd, –Ksz, –K sz not supported, silently ignored.
- –KPIC, –K PIC generate Position Independent Code (this is the default for the MIPS ABI).
- –Kminabi, –K minabi restrict libraries to ABI definitions (not needed for MIPS ABI, ignored).
- –Kxgot, –K xgot generate code to use an expanded Global Offset Table (GOT).
- –L controls library search path at linking phase.
- –l search specified library.
- –o names output file.
- –O controls optimization.
- –P perform only preprocessing.
- –p generate profiling code.
- –Qn, –Q n, –Qy, –Q y generates compiler identification in output.

- –q controls generation of profiling code.

NOTE The –**q** option is deprecated, and will not be supported or documented in future editions of the *Conformance Guide*.

- –S suppress assembly and linking (leaves assembler file).
- –U causes specified name to be undefined.
- –V prints version information.
- –W not supported, generates a warning message.
- –Xa, –X a, –Xc, –X c, –Xt, –X t controls conformance to ANSI and ISO C standards.
- –Xreadwrite_const, –X readwrite_const place *const* data in writable memory. Selecting this option is a violation of the ANSI C standard.
- –Xreadonly_const, –X readonly_const place *const* data in read-only memory. This is the default.
- –Y not supported, generates a warning message.

The `abild` wrapper supports the following flags, some of which are described in the `ld(SD_CMD)` page in the *System V Interface Definition, Third Edition (SVID 3)*.

- –a undefined references generate errors (static mode).
- –b controls DSO linkage.
- –dn, –d n, –dy, –d y selects dynamic (*y*) or static (*n*) linking.
- –e defines entry point.
- –h controls dynamic linking name lookup.
- –l search the specified library.
- –m produce a memory map.
- –o names the output file.
- –r combine object files.
- –s strip symbolic information.
- –t suppress warning about multiple defined symbols of different sizes.
- –u enters named symbol as an undefined.
- –zdefs, –z defs, –znodefs, –z nodefs, –ztext, –z text controls handling of undefined symbols.
- –Bsymbolic, –B symbolic, –Bdynamic, –B dynamic, –Bstatic, –B static control lib. name lookup
- –G produce shared object.
- –I specifies an interpreter to be linked with a.out.
- –L controls library lookup path.

- –M use specified map file.
- –Qn, –Q n, –Qy, –Q y add compiler identification strings to output.
- –V output a version identifier string.
- –Y controls library search list.

5 PROGRAM LOADING AND DYNAMIC LINKING

Dynamic Linking

NOTE This section is new as of Version 1.2 of the Conformance Guide, but will not be marked with diffmarks.

Shared Object Dependencies

The MIPS processor specific ABI defines the default library search path to be `/lib:/usr/lib:/usr/lib/cmplrs/cc`. This specification defines the default library search path to be `/lib:/usr/lib:/usr/lib/cmplrs/cc:/opt/lib`. The `/usr/lib/cmplrs/cc` portion is deprecated and will be dropped in a future version of this specification.

Program Interpreter

For MIPS ABI conforming programs, the program interpreter (the contents of the `PT_INTERP` segment) shall be `/usr/lib/libc.so.1`.

Initialization and Termination Functions

After the dynamic linker has built the process image and performed the relocations, each shared object gets the opportunity to execute some initialization code. All shared object initializations happen before the executable file gains control.

Before the initialization code for any object A is called, the initialization code for any other objects that object A depends on are called. For these purposes, an object A depends on another object B, if B appears in A's list of needed objects (recorded in the DT_NEEDED entries of the dynamic structure). The order of initialization for circular dependencies is undefined.

The initialization of objects occurs by recursing through the needed entries of each object. The initialization code for an object is invoked after the needed entries for that object have been processed. In the case that more than one object is available for initialization, reverse resolution ordering will determine the initialization order.

The example on the following page illustrates correct initialization ordering. In this example the a.out is dependent on b, d, and e. b is dependent on d and f, while d is dependent on e and g. From this information a dependency graph can be drawn. The above algorithm on initialization will then allow the following specified initialization ordering.

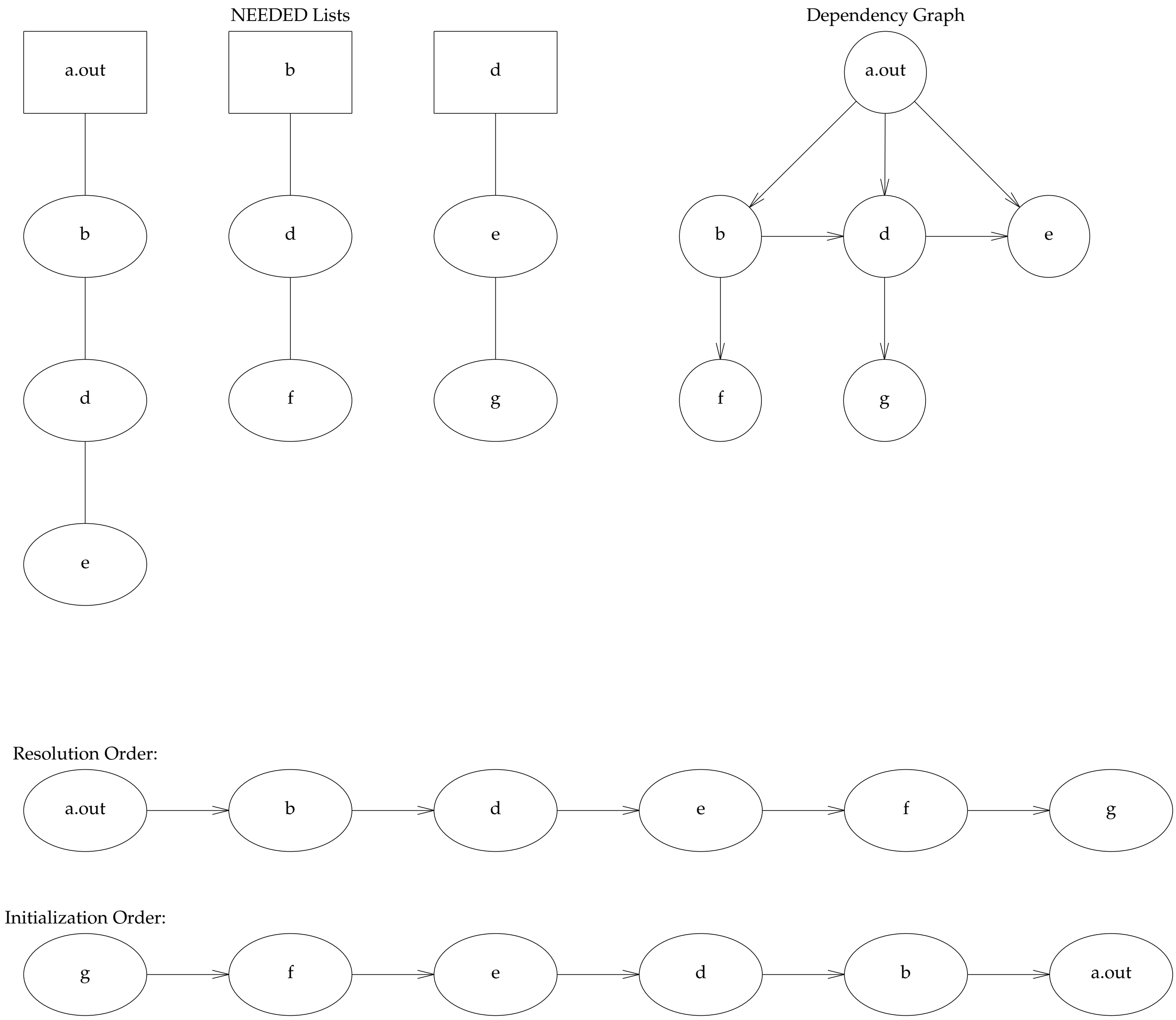
Similarly, shared objects may have termination functions, which are executed with the `atexit`(BA_OS) mechanism after the base process begins its termination sequence. The order in which the dynamic linker calls termination functions is the exact reverse order of their corresponding initialization functions. If a shared object has a termination function, but no initialization function, the termination function will execute in the order it would have as if the shared object's initialization function was present. The dynamic linker ensures that it will not execute any initialization or termination functions more than once.

Shared objects designate their initialization and termination functions through the `DT_INIT` and `DT_FINI` entries in the dynamic structure, described in ''Dynamic Section'' in the generic ABI. Typically, the code for these functions resides in the `.init` and `.fini` sections, mentioned in ''Sections'' of Chapter 4 of the generic ABI.

NOTE Although the `atexit`(BA_OS) termination processing normally will be done, it is not guaranteed to have executed upon process death. In particular, the process will not execute the termination processing if it calls `_exit` [see `exit`(BA_OS)] or if the process dies because it received a signal that it neither caught nor ignored.

The dynamic linker is not responsible for calling the executable file's `.init` section or registering the executable file's `.fini` section with `atexit`(BA_OS). Termination functions specified by users via the `atexit`(BA_OS) mechanism must be executed before any termination functions of shared objects.

Figure 5-1: Initialization Ordering Example



6 LIBRARIES

Introduction

This chapter describes additions and clarifications to the ABI library specifications. Facilities in these libraries manipulate system data files, trap to the operating system, and so on. The routines described here are corrections or omissions from the base documents, or additions specific to the MIPS ABI environment. The following areas are defined:

`libc`	The C library, containing various facilities defined by System V, ANSI C, POSIX, and so on. It contains interfaces to basic system services. Some special cases are covered in detail: `mmap` Reserved range for `MMAP_FIXED` `sysinfo` MIPS ABI specific additions `mtio` Tape Manipulation `ioctl` Calls
`libdl`	The dynamic linking library, containing routines that give the user direct access to the dynamic linking facilities.
`libnsl`	The networking services library, containing the transport layer interface routines, as well as routines for machine independent data representation (XDR), remote procedure calls (RPC) and other networking support.
`libmutex`	Routines allowing programs to access fast mutual exclusion features.
`libsocket`	Routines providing an interface to Berkeley Software Distribution (BSD) networking facilities based on the socket model.
`libabi`	A library containing assorted routines which are additions to the base ABI documents. For the current release of this specification, if contains only the Asynchronous I/O Interfaces from POSIX 1003.1b.
`libX`	A library for building applications using the X Window System, Version 11 protocol.

While not all ABI-conforming systems provide a complete ABI development environment, all are required to have the ability to link ABI-conforming objects together with locally produced objects to produce a localized binary. The actual environment to link ABI objects into a localized binary may be an optional package. Certain additional interfaces are provided for the use of ABI conforming programs in the form of of static archives which are required to be present if the the ability to link ABI objects is installed on a platform.

`libm`	Math Library
`libXm`	OSF/Motif Library
`libgen`	General-Purpose Library
`libucb`	BSD Emulation Library

Shared Library Names

Executable and shared object files contain the names of required shared libraries. The following shared libraries are required by the MIPS ABI.

Figure 6-1: Shared Library Names

Library	Reference Name
`libc`	`/usr/lib/libc.so.1`
`libnsl`	`/usr/lib/libnsl.so`
`libX`	`/usr/lib/libX11.so.2`
`libmutex`	`/usr/lib/libmutex.so`
`libdl`	`/usr/lib/libdl.so`
`libsocket`	`/usr/lib/libsocket.so`
`libabi`	`/usr/lib/libabi.so.1`

System Library

The System Library, `libsys`, as described in the generic ABI Chapter 6, has been removed from the MIPS ABI. Originally, will not necessarily exist as a separate library on ABI conforming systems. Thus, `/usr/lib/ld.so.1` is not required. As described in the generic ABI, all `libsys` symbols are present in `libc`.

C Library

The C library, `libc`, contains all the symbols listed in the generic ABI and MIPS processor specific ABI, as well as the symbols described in the following sections.

Additional Entry Points

The following routines are included in the `libc` library to provide entry points for the required source-level interfaces listed in the generic ABI. A description and syntax summary for each function follows the table.

Figure 6-2: `libc` **Additional Required Entry Points**

`_fxstat`	`_lxstat`	`_xstat`	`_xmknod`	`nuname`	`_nuname`

NOTE These entry points should not be called directly by application programs, which should continue to use the source-level interfaces `fstat`, `lstat`, `stat`, `mknod` and `uname`.

In order to implement the source-level interfaces, systems must provide stubs for the `stat` family and `mknod` in `<sys/stat.h>`, and for `uname` in `<sys/utsname.h>`. Applications, in turn, must include those files to get the ABI-specified behavior, otherwise the older interfaces (those extant before the Expansion of Fundamental Types (or EFT)) may erroneously be included.

```
int _fxstat(const int, int, struct stat *);
```
The semantics of this function are identical to those of the `fstat`(BA_OS) function describes in the *System V Interface Definition, Third Edition (SVID 3)*. Its only difference is that it requires an extra first argument whose value must be 2.

```
int _lxstat(const int, const char *, struct stat *);
```
The semantics of this function are identical to those of the `lstat`(BA_OS) function described in the *SVID 3* Its only difference is that it requires an extra first argument whose value must be 2.

```
int _xstat(const int, const char *, struct stat *);
```
The semantics of this function are identical to those of the `stat`(BA_OS) function described in the *SVID 3* Its only difference is that it requires an extra first argument whose value must be 2.

```
int _xmknod(const int, const char *, mode_t, dev_t);
```
The semantics of this function are identical to those of the `mknod`(BA_OS) function described in the *SVID 3* Its only difference is that it requires an extra first argument whose value must be 2.

```
int nuname(struct utsname *);
```
The semantics of this function are identical to those of the `uname`(BA_OS) function described in the *SVID 3* The symbol `_nuname` is also available with the same semantics.

Support Routines

Figure 6-3: `libc` **Contents, Names without Synonyms**

`_cleanup`	`_numeric`	`modf`

Figure 6-4: `libc` **Contents, Names with Synonyms**

`getitimer`	`gettimeofday`	`logb`	`makecontext`
`nextafter`	`sbrk*`	`scalb`	`select`
`setitimer`	`setlabel`	`setegid*`	`seteuid*`
`swapcontext`	`sysinfo`		

*Function is new to version 1.2

Note that the following routines are listed in the generic ABI as Names without Synonyms.

Figure 6-5: `libc` **Contents, Names with Synonyms, previously listed without Synonyms**

`fopen`	`vfprintf`	`vprintf`	`vsprintf`

`void *sbrk(int incr)`

The `sbrk` function is used to grow or shrink the area of a process which can be used for data storage. If *incr* is positive, it will grow the area, if negative, it will shrink it. If the data area is expanded past the maximum previous size, the new area will be initialized to zero, but the contents of any space which had been previously allocated will be undefined.

`sbrk` returns a pointer to the previous end of the data area or `−1` if it was unable to change the size as requested. If `−1` is returned, the data storage area will remain the previous size and `errno` will be set to one of the following:

`ENOMEM` Allocating the requested space would exceed the maximum amount of space allowed to the process.

`EAGAIN` The allocation could have succeeded but there is currently not enough memory available in the system to satisfy the request.

`int setegid (gid_t group)`

`int seteuid (uid_t user)`

`seteuid` and `setegid` set the effective user ID and group ID of the current process.

If the effective user ID of the calling process is not super-user, but if its real user (group) ID is equal to *uid* (*gid*), or if the saved set-user (group) ID from `execve` is equal to *uid* (*gid*), then the effective user (group) ID is set to *uid* (*gid*).

These functions return 0 on success, -1 on failure and set `errno` to indicate the error.

`long sysinfo (int command, char *buf, long count)`

`sysinfo` copies information relating to the system on which the process is executing into the buffer pointed to by *buf*. Upon successful completion, the value returned indicates the buffer size in bytes required to hold the complete value and the terminating null character. If this value is no greater than the value passed in *count*, the entire string was copied; if this value is greater than *count*, the string copied into *buf* has been truncated to *count-1* bytes plus a terminating null character.

Otherwise, a value of `-1` is returns and *errno* is set to indicate the error. `sysinfo` can be called with a *count* of `0` to determine the required buffer size. Alternatively, a reasonable guess for *count* is 256, which is likely to cover all strings returned by this interface in typical installations.

NOTE Operating system implementors should note that the SVR4 definition of `gettimeofday` differs from the BSD4.3 version.

`mmap` Routine

A group of programs may need to specify the address of shared memory and mapped file regions in order to use the same addresses to access these regions. The address range from `0x30000000` to `0x3fff0000` is reserved for this purpose; additional contiguous space above this range is recommended and may be supported. A MIPS ABI conforming system guarantees that shared memory and mapped file regions placed by the system will not fall in this range.

The placement of a shared memory or mapped file region by a program must follow these rules:

- The address range of the reserved region will at least lie between `0x30000000-0x3fff0000`.
- No two fixed regions may occupy the same chunk of memory, modulo 256K.
- The address of a mapped file region and the corresponding file offset must be congruent modulo 256K.

For example, if a program wants to create a `0x3000`-byte shared memory region and map a file with an offset of `0x2000` and length of `0x1000`, both regions will have fixed addresses. One possible placement of these regions puts the shared memory region at `[0x30000000,0x30002fff]` and `[0x30042000,0x30042fff]`.

NOTE The above discussion only applies when fixed addresses must be used (`MAP_FIXED` is used in *flags*). As noted in the documentation for `mmap`, the use of `MAP_FIXED` is discouraged for general use.

NOTE Implementors using `mmap` are advised that the generic ABI requires

- the system will zero fill any partial page at the end of an `mmap`'ed object
- the system will never write out any modified portions of the last page of an object which are beyond its end, i.e. `mmap` cannot be used to extend the size of a file.
- references to whole pages following the end of an object will result in the delivery of a `SIGBUS` signal.

sysinfo Routine

The following are additions to commands supported by sysinfo. The new and recommended routines are described below, along with examples. A sysinfo header file, including the additional commands, follows. See vendor release notes for vendor values.

_MIPS_SI_VENDOR — Copy into the array pointed to by *buf* a string which is the name of the company providing the system. The vendor is the company actually selling the product rather than the company which built the product.

Example: Silicon Graphics, Inc.

_MIPS_SI_OS_PROVIDER — Copy into the array pointed to by *buf* a string which is the name of the company providing the operating system. This is the name of the company which built the operating system.

Example: Silicon Graphics, Inc.

_MIPS_SI_OS_NAME — Copy into the array pointed to by *buf* a string which is the name of the operating system.

Example: IRIX

_MIPS_SI_HW_NAME — Copy into the array pointed to by *buf* a string which is the name of the hardware platform. The name is vendor selected.

Example: IP12

_MIPS_SI_NUM_PROCESSORS — Copy into the array pointed to by *buf* a string which is the ASCII representation of the number of processors. This should be the number of currently active, available processors.

NOTE Application developers should use this value cautiously. In some systems, this value does not reflect processors which have been previously allocated to other tasks and are unavailable for further allocation. The new (as of Version 1.2 of this document) _MIPS_SI_AVAIL_PROCESSORS request should be used in place of this request as it provides an unambiguous report on the number of processors available for assignment.

_MIPS_SI_HOSTID — Copy into the array pointed to by *buf* a string which is the ASCII representation of the host id in hex.

Example: c052d008

NOTE OS implementors: this value is intended to be used by license management software. The number can be generated from any appropriate source as long as it is guaranteed that it would be difficult for a system administrator to change the value only to circumvent node-lock type license restrictions. In particular, the IP address may not be appropriate, as it is easily changed.

_MIPS_SI_OSREL_MAJ — Copy into the array pointed to by *buf* a string which is the ASCII representation of the OS major release number.

`_MIPS_SI_OSREL_MIN` Copy into the array pointed to by *buf* a string which is the ASCII representation of the OS minor release number.

`_MIPS_SI_OSREL_PATCH` Copy into the array pointed to by *buf* a string which is the ASCII representation of the OS patch release number.

Release patch (5.1.1) is an example:

```
_MIPS_SI_OSREL_MAJ:   5
_MIPS_SI_OSREL_MIN:   .1.1
_MIPS_SI_OSREL_PATCH  0
```

`_MIPS_SI_PROCESSORS` Copy into the array pointed to by *buf* a string which is processor type. In the case of a multiprocessor, this will be a comma separated list with `_MIPS_SI_NUM_PROCESSORS` entries.

NOTE OS implementors: The following is the mapping from the value returned by the chip's "imp" register to the processor type string which should be returned.

imp	processor name
0	R2000
1	R2000
2	R3000A
3	R6000
4	R4000, R4400
6	R6000A
0x9	R10000 (T5)
0xA	R4200
0xB	R4300
0x10	R8000
0x20	R4600

Example: `R2000A/R3000 3.0`

For the R4400, the chip's "rev" register is 40 so that the imp/rev pair reads 0x440, while for the R4000 it reads 0x400.

`_MIPS_SI_AVAIL_PROCESSORS` Copy into the array pointed to by *buf* a string which is the ASCII representation of the number of processors which are currently active and available for general use, i.e. are not reserved for some specific task. In most cases `_MIPS_SI_AVAIL_PROCESSORS` will be equal to `_MIPS_SI_NUM_PROCESSORS`, but some system implementations may permit processors to be dedicated to specific tasks such as a real-time environment, and they would be unavailable for general purpose assignment. In such a case `_MIPS_SI_AVAIL_PROCESSORS` would be less than `_MIPS_SI_NUM_PROCESSORS`. In no case shall the OS return a value for `_MIPS_SI_AVAIL_PROCESSORS`which is larger than `_MIPS_SI_NUM_PROCESSORS`.

NOTE

There are no additions to the system level commands.

Figure 6-6: Header File: `<sys/systeminfo.h>`

```
#define SI_SYSNAME              1
#define SI_HOSTNAME             2
#define SI_RELEASE              3
#define SI_VERSION              4
#define SI_MACHINE              5
#define SI_ARCHITECTURE         6
#define SI_HW_SERIAL            7
#define SI_HW_PROVIDER          8
#define SI_SRPC_DOMAIN          9
#define _MIPS_SI_VENDOR         100
#define _MIPS_SI_OS_PROVIDER    101
#define _MIPS_SI_OS_NAME        102
#define _MIPS_SI_HW_NAME        103
#define _MIPS_SI_NUM_PROCESSORS 104
#define _MIPS_SI_HOSTID         105
#define _MIPS_SI_OSREL_MAJ      106
#define _MIPS_SI_OSREL_MIN      107
#define _MIPS_SI_OSREL_PATCH    108
#define _MIPS_SI_PROCESSORS     109
#define _MIPS_SI_AVAIL_PROCESSORS 110
```

Tape Manipulation `ioctl` Calls

NOTE

The entire `mtio` section is new as of Version 1.2 of the Conformance Guide, but will not be marked with diffmarks.

A standard set of binary compatible `ioctl` calls for the postitioning and movement of tape devices is part of the ABI with Version 1.2 of this document. As with other aspects of the ABI, the intent of these definitions is to allow a machine independent mechanism of tape positioning and manipulation.

As these specifications were discussed, some fundamental inconsistencies were discovered among the existing implementations of the tape manipulation `ioctl`'s. Because each vendor must retain binary compatibility for their own software, new values have been established for these interfaces, rather than reusing existing values.

The header file `<sys/mtio.h>` (described in the *System V Interface Definition, Third Edition*. but not formally part of the generic ABI) shall include the following additional definitions.

Figure 6-7: Additions to Header File: `<sys/mtio.h>`

```
#define ABI_MTIOCTOP (('t' << 8) | 9)

struct abi_mdata {
        short   mt_op;          /* Operation, see below */
        long    mt_cnt;         /* Operation count */
        unsigned short mt_sts;  /* Returned status, see below */
        long    reserved1;
        long    reserved2;
        long    reserved3;
};
#define ABI_MTWEOF      0       /* Write tape mark */
#define ABI_MTFSF       1       /* Forward space file
#define ABI_MTBSF       2       /* Backward space file
#define ABI_MTFSR       3       /* Forward space record
#define ABI_MTBSR       4       /* Backward space record
#define ABI_MTREW       5       /* Rewind
#define ABI_MTOFFL      6       /* Rewind or Rewind and Unload
```

`ABI_MTIOCTOP` is equivalent to the hex value `0x7409`.

The value of the *mt_sts* element on return is specific to different tape devices and some implementations may not modify this element.

The fields *reserved1*, *reserved2* and *reserved3* are included for compatibility.

`ABI_MTWEOF` This operation writes file mark at the current tape location. A file mark is a delimiter containing no data. When the tape is positioned immediately before a file mark, a `read` function call will return a value of zero. This operation is not considered as writing data. Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_cnt	Input	Number of file marks to write
mt_sts	Output	Result status

`ABI_MTFSF` This operation moves the tape forward until *mt_cnt* file marks have been passed, leaving it positioned after the last file mark. Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_cnt	Input	Number of file marks to space forward.
mt_sts	Output	Resulting status.

`ABI_MTBSF` This command is similar to the `ABI_MTFSF` except that the direction of tape movement is reversed and the tape will be positioned before the file mark. The use of *abi_mdata* fields is identical. Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_cnt	Input	Number of file marks to space backward.
mt_sts	Output	Resulting status.

This command may be unsupported on some tape devices.

`ABI_MTFSR`

This operation spaces the tape forward a specified number of records. Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_cnt	Input	Number of records to space forward.
mt_sts	Output	Resulting status.

`ABI_MTBSR`

This command is similar to the `ABI_MTFSR` command except that the direction of tape movement is reversed. The use of *abi_mdata* fields is identical.

Name	Input or Output	Description
mt_cnt	Input	Number of records to space forward.
mt_sts	Output	Resulting status.

This command may be unsupported on some tape devices.

`ABI_MTREW`

This operation rewinds the tape to the beginning, but it remains on-line. Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_sts	Output	Resulting status.

`ABI_MTOFFL`

This operation rewinds the tape to the beginning, if it is not already at that position. It then does any device-specific actions required to unload the tape. On some magnetic tape devices, unloading may be entirely manual and so this is identical in action to `ABI_MTREW.` Fields used in *abi_mdata* are:

Name	Input or Output	Description
mt_sts	Output	Resulting status.

Tape Positioning

The position of the tape read/write head must be consistent across platforms for applications using tape `ioctl` calls. The physical data on tapes varies between different device types and between driver implementations, but there is a consistent interpretation of the logical sequence of information on a tape as seen with the `read` function. Assume the following logical information on a tape, with the beginning of tape to the left and the end of the tape to the right and positions labeled 0 through 5.

Figure 6-8: Example Tape Layout

```
<data1>   <data2>   <FM1>   <data3>   <FM2>   <EOD>
^         ^         ^       ^         ^       ^
|0        |1        |2      |3        |4      |5
```

Each labelled position is to the beginning-of-tape side of the data blocks and file marks. Attempts to read the end-of-data (EOD) information will receive either an error indication or zero bytes read, depending upon the implementation.

Sequential single-block `read` calls from the beginning of the tape in this example will return <data1>, <data2>, end-of-file (indicated by a return value of zero), <data 3>, end-of-file and end-of-data. The data blocks in this example are all the same size. Detection of end-of-data on the tape is indicated by a `read` with a return value of zero, followed by a `read` which returns zero or an error indication with *errno* set to `ENOSPC.`

Each position is such that if a `read` is performed, the block beginning at that position will be read, i.e. if the tape is at position 1, a `read` will return the contents of block "data2". The following table summarizes the final position of the tape head when various commands are performed with the tape head at a given starting position, a value of *mt_cnt* of one and with successful (no error) completion of the `ioctl` call:

Figure 6-9: Tape Positioning after `ioctl` Operations

Starting	Command				
Position	FSF	BSF	FSR	BSR	`read`
0	3	-	1	-	1
1	3	-	2	0	2
2	3	-	-	1	3 w/ EOF indication
3	5	-	4	-	4
4	5	2	-	3	5 w/ EOF indication
5	-	4	-	-	undefined

NOTE Tape manipulation has proven to be an extremely subtle issue. The discussions of the exact semantics of the various operations have highlighted the fact that many different interpretations of the basic semantics are possible. Operating system implementors should pay careful attention to the semantics expressed in the above table, as they define the ABI behavior. It is believed that these semantics are compatible with most of the existing implementations.

The `read` indicated above is for the size of one data block. The EOF indication for `read` is a return value of zero.

Items in the table indicated by a hyphen result in a tape position which is not defined by the ABI. It is implementation-dependent whether an error condition is reported by the `ioctl` call.

Positioning on Close

The tape position after closing on devices opened with rewind access is the beginning of the tape. The position after closing on devices opened with no-rewind access is generally undefined. However, if a file mark has just been read the tape will remain at the current position when closed.

If the last operation was to write data, a file mark is written. The applies whether the device was opened with rewind or no-rewind access. Some tape drives write an end of data marker at the firmware level. The tape position after closing on devices opened with no-rewind access will be after the file mark.

Figure 6-10: Tape Positioning after `close` for device opened with no-rewind access

`Last Operation`	Action After `close`
`write`	Write `EOF`, position after `EOF`.
`read which returned EOD indication`	Position after `EOF`
`ioctl(ABI_MTWEOF)`	Position after `EOF`
`ioctl(ABI_MTFSF)`	Leaves positioned after `EOF`
All others	Undefined

Effects of `write`

Data written with a `write` function will be written starting at the first data block or file mark following the current tape position. The effect on data following the written data is undefined and may not be retrievable by subsequent tape positioning and `read` functions.

If the tape has just been positioned to the point after a file mark, writing to the tape will start a new tape file even if the file mark is the last one prior to the end-of-data. This allows new tape files to be added to the end of the previous data.

Return Values

The `ioctl` function returns `–1` when errors occur and sets the global variable *errno* to error codes defined in `<errno.h>`. The values returned in the *mt_sts* element of the *abi_mdata* structure are device-specific and should only be used for display purposes.

Reading an end-of-file causes a return value of zero from the `read` call. If the file mark read is the last one on the tape, a subsequent `read` will return either zero or an error indication with *errno* set to `ENOSPC`.

Partitioned Tapes

Some magnetic media are logically partitioned into multiple logical tapes. When the commands specified herein are used on such a medium they apply to the logical tape rather than the physical tape. For example, rewind media command for a partitioned 4mm DDS tape will position the write head before the first block of the current partition, which may be different than the first block of the physical medium. The specification of commands which affect media partitioning is outside the scope of the ABI.

Runtime Dynamic Linking Library

The libdl library allows users to dynamically link shared libraries after a program is launched. The library contains the following entry points.

Figure 6-11: libdl Contents

dlopen	dlclose	dlsym	dlerror

Shared libraries are accessed with dlopen and released with dlclose. Symbols in shared libraries opened via dlopen can be located by calling dlsym. libdl is implemented as a shared library, linked on the command line using -ldl.

`void *dlopen(char *pathname, int mode);`
This routine makes a shared object available to a running process. It returns a handle for future dlclose and dlsym calls.

`int dlclose(void *handle);`
Disassociates a shared object previously opened by dlopen from the current process.

`void *dlsym(void *handle, char *name);`
Allows a process to obtain the address of a symbol defined within a shared object previously opened by dlopen.

`char *dlerror(void);`
Returns a formatted character string describing the last error that occurred during dynamic linking processing.

Implementation notes:

- dlopen looks for DT_NEEDED entries in the .dynamic section of the shared library it opens and, for each entry, invokes dlopen on the shared library entry names. This process recurses until all libraries specified by DT_NEEDED entries have been loaded. In no case will the dynamic linker link in a shared library twice.
- Normally, the exported symbols from an object accessed with dlopen are directly available only to those other objects that were loaded as a result of the same call to dlopen. If the value RTLD_GLOBAL (which is defined to be 4) is or'ed with the mode argument, the exported symbols of all objects loaded via this call to dlopen are directly available to all other dlopen'd objects. This behavior is part of the behavior specified in a future edition of the generic ABI (the 3rd edition).
- dlclose unmaps the shared library associated with the handle passed to it and any other libraries that were linked via the recursive process described above. It guarantees that a library is not unlinked until all the references to it are closed.
- dlsym restricts its search to the shared libraries linked by the dlopen that created its handle. The search looks first in the shared library referenced by the handle and then recursively traverses libraries named in DT_NEEDED entries in a breadth-first order.
- dlerror returns an English message; it is not internationalized.

Networking Services Library

The Networking Services library, `libnsl`, contains all the symbols listed in the generic ABI and MIPS processor specific ABI, as well as the following symbols, which were omitted from the base documents or were not fully defined.

Additional Entry Points

The following routines are included in `libnsl` to correct oversights in the base documents.

Figure 6-12: `libnsl` **Contents, Additional Required Entry Points**

```
nc_sperror*          netdir_perror*    netdir_sperror*   xdr_u_int*
xdrrec_endofrecord*  xdrrec_skiprecord*
```

*Function is new to version 1.2

These routines are defined in the SVID 4th Edition.

Figure 6-13: `libnsl` **Contents, Global External Data Symbols**

```
_nderror*            svc_fdset*        t_errno*          rpc_createerr*
```

*Symbol is new to version 1.2

Mutual Exclusion Library

The Mutual Exclusion Library, libmutex, provides ABI entry points for mutual exclusion.

Figure 6-14: libmutex **Contents**

init_lock	acquire_lock	release_lock	spin_lock*
stat_lock			

*Function is new to version 1.2

These functions supply a user-level interface to mutual exclusion. The parameter lck must point to memory shared by all processes wishing to acquire or test the lock. The library routines might not contain checks for the validity (e.g. non-null value) of the lck parameter. If only a single process will be using the lock, lck need not reside in shared memory. libmutex is provided as a shared library, linked on the command line using -lmutex. The header file <abi_mutex.h> should be included.

int init_lock(abilock_t *lck);
: init_lock must be called on a lock before any of the other functions. It initializes the lock to an unlocked state. A non-zero return status indicates an error.

int acquire_lock(abilock_t *lck);
: acquire_lock tries once to acquire the lock referenced by lck. It returns zero if the lock was acquired, otherwise non-zero.

int release_lock(abilock_t *lck);
: release_lock unconditionally releases the lock pointed to by lck. The ability for one process to release the lock of another process is permitted. A non-zero return status will indicate an error.

void spin_lock(abilock_t *lck);
: spin_lock tries to acquire the lock referenced by lck and does not return until the lock is acquired.

int stat_lock(abilock_t *lck);
: stat_lock returns the current state of the lock referenced by lck without attempting to acquire the lock. It returns zero if the lock is free, otherwise non-zero.

The contents of the structure abilock_t are defined in <abi_mutex.h> as follows:

Figure 6-15: Header File: `<abi_mutex.h>`

```
int init_lock(abilock_t *lck);
int acquire_lock(abilock_t *lck);
int release_lock(abilock_t *lck);
void spin_lock(abilock_t *lck);
int stat_lock(abilock_t *lck);

typedef struct {
   unsigned long    abi_lock;
} abilock_t;
```

NOTE

Locks may not be freed automatically by the system. Locks acquired by a process that dies may remain locked.

The user is responsible for implementing a back-off policy if `acquire_lock` returns failure.

Even though `stat_lock` returns status indicating that the lock is available, a call to `acquire_lock` could still fail.

Although the name `spin_lock` implies that the implementation must "busy wait" by repeatedly trying to acquire the lock, this is not necessarily the best implementation. OS developers are free to implement more sophisticated schemes provided that the intended semantics of the call are maintained.

Socket Library

This definition is derived from the SVR4 specification of sockets, which depends on the Networking Services Library (`libnsl`) for services. However, applications that rely on assumptions about the underlying implementation of sockets may not be ABI compliant. In particular, stream operations on sockets are not supported. Constants labeled "may not be supported" are included for consistency between vendors but may not be implemented on all ABI compliant platforms. However, the constants are required to be included in the header files. The socket library is supplied as a shared library, `libsocket.so`. The socket library (`-lsocket`) must precede the networking services library (`-lnsl`) in the `cc` command line.

NOTE The behavior of a zero queue length (the second argument) to the `libsocket` routine `listen` is undefined.

Figure 6-16: `libsocket` **Contents**

accept	bind	connect
endhostent	endnetent	endprotoent
endservent	ether_aton	ether_hostton
ether_line	ether_ntoa	ether_ntohost
gethostbyaddr	gethostbyname	gethostent
getnetbyaddr	getnetbyname	getnetent
getpeername	getprotobyname	getprotobynumber
getprotoent	getservbyname	getservbyport
getservent	getsockname	getsockopt
htonl	htons	inet_addr
inet_lnaof	inet_makeaddr	inet_netof
inet_network	inet_ntoa	listen
ntohl	ntohs	recv
recvfrom	recvmsg	send
sendmsg	sendto	sethostent
setnetent	setprotoent	setservent
setsockopt	shutdown	socket
socketpair		

Data Definitions for Sockets

The following system header files contain the data structures and manifest constants required for sockets.

Figure 6-17: Header File: `<net/if.h>`

```
#define IFF_UP           0x1
#define IFF_BROADCAST    0x2
#define IFF_DEBUG        0x4
#define IFF_LOOPBACK     0x8
#define IFF_POINTOPOINT  0x10
#define IFF_NOTRAILERS   0x20
#define IFF_RUNNING      0x40
#define IFF_NOARP        0x80
#define IFF_PROMISC      0x100
#define IFF_ALLMULTI     0x200
#define IFF_INTELLIGENT  0x400
#define IFF_MULTICAST    0x800
#define IFF_PRIVATE      0x8000

struct   ifreq {
#define   IFNAMSIZ   16
   char   ifr_name[IFNAMSIZ];
   union {
      struct  sockaddr ifru_addr;
      struct  sockaddr ifru_dstaddr;
      char    ifru_oname[IFNAMSIZ];
      struct  sockaddr ifru_broadaddr;
      short   ifru_flags;
      int     ifru_metric;
      char    ifru_data[1];
      char    ifru_enaddr[6];
   } ifr_ifru;
#define   ifr_addr      ifr_ifru.ifru_addr
#define   ifr_dstaddr   ifr_ifru.ifru_dstaddr
#define   ifr_oname     ifr_ifru.ifru_oname
#define   ifr_broadaddr ifr_ifru.ifru_broadaddr
#define   ifr_flags     ifr_ifru.ifru_flags
#define   ifr_metric    ifr_ifru.ifru_metric
#define   ifr_data      ifr_ifru.ifru_data
#define   ifr_enaddr    ifr_ifru.ifru_enaddr
};

struct   ifconf {
   int   ifc_len;
   union {
      caddr_t   ifcu_buf;
      struct    ifreq *ifcu_req;
   } ifc_ifcu;
#define   ifc_buf   ifc_ifcu.ifcu_buf
#define   ifc_req   ifc_ifcu.ifcu_req
};
```

Figure 6-18: Header File: `<net/if_arp.h>`

```
struct arpreq {
   struct   sockaddr arp_pa;
   struct   sockaddr arp_ha;
   int      arp_flags;
};

#define   ATF_INUSE        0x01
#define   ATF_COM          0x02
#define   ATF_PERM         0x04
#define   ATF_PUBL         0x08
#define   ATF_USETRAILERS  0x10
```

Figure 6-19: Header File: `<netinet/if_ether.h>`

```
typedef u_char ether_addr_t[6];
```

Figure 6-20: Header File: <netinet/in.h>

```
#define ntohl(x) (x)
#define ntohs(x) (x)
#define htonl(x) (x)
#define htons(x) (x)

struct in_addr {
   u_long  s_addr;
};

#define INADDR_ANY (u_long)0x00000000

struct sockaddr_in {
   short   sin_family;
   u_short sin_port;
   struct  in_addr sin_addr;
   char    sin_zero[8];
};

#define IPPROTO_IP    0
#define IPPROTO_ICMP  1
#define IPPROTO_IGMP  2
#define IPPROTO_GGP   3
#define IPPROTO_TCP   6
#define IPPROTO_EGP   8
#define IPPROTO_PUP   12
#define IPPROTO_UDP   17
#define IPPROTO_IDP   22
#define IPPROTO_HELLO 63
#define IPPROTO_ND    77

#define IPPROTO_RAW   255
#define IPPROTO_MAX   256

#define IP_OPTIONS          1
#define IP_MULTICAST_IF     2   /* may not be supported */
#define IP_MULTICAST_TTL    3   /* may not be supported */
#define IP_MULTICAST_LOOP   4   /* may not be supported */
#define IP_ADD_MEMBERSHIP   5   /* may not be supported */
#define IP_DROP_MEMBERSHIP  6   /* may not be supported */
#define IP_HDRINCL          7   /* may not be supported */
#define IP_TOS              8   /* may not be supported */
#define IP_TTL              9   /* may not be supported */
#define IP_RECVOPTS         10  /* may not be supported */
#define IP_RECVRETOPTS      11  /* may not be supported */
#define IP_RECVDSTADDR      12  /* may not be supported */
#define IP_RETOPTS          13  /* may not be supported */
```

Figure 6-21: Header File: `<netinet/tcp.h>`

```
#define TCP_NODELAY     0x01
#define TCP_MAXSEG      0x02
```

Figure 6-22: Header File: `<sys/ioctl.h>`

```
#define   IOCPARM_MASK  0xff
#define   IOC_VOID      0x20000000
#define   IOC_OUT       0x40000000
#define   IOC_IN        0x80000000
#define   IOC_INOUT     (IOC_IN|IOC_OUT)
#define   _IO(x,y)      (IOC_VOID|(x<<8)|y)
#define   _IOR(x,y,t)   (IOC_OUT|((((int)sizeof(t))\
   &IOCPARM_MASK)<<16)|(x<<8)|y)
#define   _IORN(x,y,t)  (IOC_OUT|(((t)\
   &IOCPARM_MASK)<<16)|(x<<8)|y)
#define   _IOW(x,y,t)   (IOC_IN|((((int)sizeof(t))\
   &IOCPARM_MASK)<<16)|(x<<8)|y)
#define   _IOWN(x,y,t)  (IOC_IN|(((t)\
   &IOCPARM_MASK)<<16)|(x<<8)|y)
#define   _IOWR(x,y,t)  (IOC_INOUT|((((int)sizeof(t))\
   &IOCPARM_MASK)<<16)|(x<<8)|y)

#define   SIOCSHIWAT     _IOW('s',  0, int)
#define   SIOCGHIWAT     _IOR('s',  1, int)
#define   SIOCSLOWAT     _IOW('s',  2, int)
#define   SIOCGLOWAT     _IOR('s',  3, int)
#define   SIOCATMARK     _IOR('s',  7, int)
#define   SIOCSPGRP      _IOW('s',  8, int)
#define   SIOCGPGRP      _IOR('s',  9, int)
#define   SIOCADDRT      _IOW('r', 10, struct rtentry)
#define   SIOCDELRT      _IOW('r', 11, struct rtentry)
#define   SIOCSIFADDR    _IOW('i', 12, struct ifreq)
#define   SIOCGIFADDR    _IOWR('i',13, struct ifreq)
#define   SIOCSIFDSTADDR _IOW('i', 14, struct ifreq)
#define   SIOCGIFDSTADDR _IOWR('i',15, struct ifreq)
#define   SIOCSIFFLAGS   _IOW('i', 16, struct ifreq)
#define   SIOCGIFFLAGS   _IOWR('i',17, struct ifreq)
#define   SIOCSIFMEM     _IOW('i', 18, struct ifreq)
#define   SIOCGIFMEM     _IOWR('i',19, struct ifreq)
#define   SIOCGIFCONF    _IOWR('i',20, struct ifconf)
#define   SIOCSIFMTU     _IOW('i', 21, struct ifreq)
#define   SIOCGIFMTU     _IOWR('i',22, struct ifreq)
#define   SIOCGIFBRDADDR _IOWR('i',23, struct ifreq)
#define   SIOCSIFBRDADDR _IOW('i',24, struct ifreq)
#define   SIOCGIFNETMASK _IOWR('i',25, struct ifreq)
#define   SIOCSIFNETMASK _IOW('i',26, struct ifreq)
```

(continued on next page)

Figure 6-22: Header File: `<sys/ioctl.h>` (continued)

```
#define   SIOCGIFMETRIC  _IOWR('i',27, struct ifreq)
#define   SIOCSIFMETRIC  _IOW('i',28, struct ifreq)
#define   SIOCSARP       _IOW('i', 30, struct arpreq)
#define   SIOCGARP       _IOWR('i',31, struct arpreq)
#define   SIOCDARP       _IOW('i', 32, struct arpreq)
#define   SIOCUPPER      _IOW('i', 40, struct ifreq)
#define   SIOCLOWER      _IOW('i', 41, struct ifreq)
#define   SIOCSETSYNC    _IOW('i',  44, struct ifreq)
#define   SIOCGETSYNC    _IOWR('i', 45, struct ifreq)
#define   SIOCSSDSTATS   _IOWR('i', 46, struct ifreq)
#define   SIOCSSESTATS   _IOWR('i', 47, struct ifreq)
#define   SIOCSPROMISC   _IOW('i', 48, int)
#define   SIOCADDMULTI   _IOW('i', 49, struct ifreq)
#define   SIOCDELMULTI   _IOW('i', 50, struct ifreq)
#define   SIOCPROTO      _IOW('s', 51, struct socknewproto)
#define   SIOCGETNAME    _IOR('s', 52, struct sockaddr)
#define   SIOCGETPEER    _IOR('s', 53, struct sockaddr)
#define   IF_UNITSEL     _IOW('s', 54, int)
#define   SIOCXPROTO     _IO('s', 55)
#define   SIOCIFDETACH   _IOW('i', 56, struct ifreq)
#define   SIOCGENPSTATS  _IOWR('i', 57, struct ifreq)
#define   SIOCX25XMT     _IOWR('i', 59, struct ifreq)
#define   SIOCX25RCV     _IOWR('i', 60, struct ifreq)
#define   SIOCX25TBL     _IOWR('i', 61, struct ifreq)
#define   SIOCSLGETREQ   _IOWR('i', 71, struct ifreq)
#define   SIOCSLSTAT     _IOW('i', 72, struct ifreq)
#define   SIOCSIFNAME    _IOW('i', 73, struct ifreq)
#define   SIOCGENADDR    _IOWR('i', 85, struct ifreq)
#define   SIOCSOCKSYS    _IOW('i', 86, struct socksysreq)
```

Figure 6-23: Header File: `<netdb.h>`

```
struct hostent {
   char   *h_name;
   char   **h_aliases;
   int    h_addrtype;
   int    h_length;
   char   **h_addr_list;
#define   h_addr   h_addr_list[0]
};

struct netent {
   char           *n_name;
   char           **n_aliases;
   int            n_addrtype;
   unsigned long  n_net;
};

struct protoent {
   char  *p_name;
   char  **p_aliases;
   int   p_proto;
};

struct servent {
   char   *s_name;
   char   **s_aliases;
   int    s_port;
   char   *s_proto;
};
```

Figure 6-24: Header File: `<net/route.h>`

```
struct rtentry {
   u_long   rt_hash;
   struct   sockaddr rt_dst;
   struct   sockaddr rt_gateway;
   short    rt_flags;
   short    rt_refcnt;
   u_long   rt_use;
   union {
      struct  ip_provider *rtu_prov;
      struct  ifnet *rtu_ifp;
   } rt_u;
#define rt_prov rt_u.rtu_prov
#define rt_ifp rt_u.rtu_ifp
};

#define RTF_UP         0x1
#define RTF_GATEWAY    0x2
#define RTF_HOST       0x4
#define RTF_REINSTATE  0x8
#define RTF_DYNAMIC    0x10
#define RTF_MODIFIED   0x20

#define RTF_SWITCHED   0x40
#define RTF_SLAVE      0x80
#define RTF_REMOTE     0x100
#define RTF_TOSWITCH   0x200
```

Figure 6-25: Header File: `<sys/socket.h>`

```
#define  AF_UNSPEC     0
#define  AF_UNIX       1
#define  AF_INET       2
#define  AF_IMPLINK    3
#define  AF_PUP        4
#define  AF_CHAOS      5
#define  AF_NS         6
#define  AF_NBS        7
#define  AF_ECMA       8
#define  AF_DATAKIT    9
#define  AF_CCITT      10
#define  AF_SNA        11
#define  AF_DECnet     12
#define  AF_DLI        13
#define  AF_LAT        14
#define  AF_HYLINK     15
#define  AF_APPLETALK  16
```

(continued on next page)

Figure 6-25: Header File: `<sys/socket.h>` (continued)

```
#define  AF_NIT        17
#define  AF_802        18
#define  AF_OSI        19
#define  AF_X25        20
#define  AF_OSINET     21
#define  AF_GOSIP      22

struct sockaddr {
   u_short  sa_family;
   char     sa_data[14];
};

#define  SO_DEBUG       0x0001
#define  SO_ACCEPTCONN  0x0002
#define  SO_REUSEADDR   0x0004
#define  SO_KEEPALIVE   0x0008
#define  SO_DONTROUTE   0x0010
#define  SO_BROADCAST   0x0020
#define  SO_USELOOPBACK 0x0040
#define  SO_LINGER      0x0080
#define  SO_OOBINLINE   0x0100
#define  SO_IMASOCKET   0x0400

#define  SOL_SOCKET   0xffff

struct   linger {
   int   l_onoff;
   int   l_linger;
};

#define  SOCK_DGRAM     1
#define  SOCK_STREAM    2
#define  SOCK_RAW       4
#define  SOCK_RDM       5
#define  SOCK_SEQPACKET 6

#define  PF_UNSPEC     AF_UNSPEC
#define  PF_UNIX       AF_UNIX
#define  PF_INET       AF_INET
#define  PF_IMPLINK    AF_IMPLINK
#define  PF_PUP        AF_PUP
#define  PF_CHAOS      AF_CHAOS
#define  PF_NS         AF_NS
#define  PF_NBS        AF_NBS
#define  PF_ECMA       AF_ECMA
#define  PF_DATAKIT    AF_DATAKIT
#define  PF_CCITT      AF_CCITT
#define  PF_SNA        AF_SNA
#define  PF_DECnet     AF_DECnet
#define  PF_DLI        AF_DLI
#define  PF_LAT        AF_LAT
#define  PF_HYLINK     AF_HYLINK
#define  PF_APPLETALK  AF_APPLETALK
```

(continued on next page)

Figure 6-25: Header File: `<sys/socket.h>` (continued)

```
#define  PF_NIT        AF_NIT
#define  PF_802        AF_802
#define  PF_OSI        AF_OSI
#define  PF_X25        AF_X25
#define  PF_OSINET     AF_OSINET
#define  PF_GOSIP      AF_GOSIP

struct msghdr {
   caddr_t  msg_name;
   int      msg_namelen;
   struct   iovec *msg_iov;
   int      msg_iovlen;
   caddr_t  msg_accrights;
   int      msg_accrightslen;
};

#define  MSG_OOB       0x1
#define  MSG_PEEK      0x2
#define  MSG_DONTROUTE 0x4

struct socknewproto {
   int     family;
   int     type;
   int     proto;
   dev_t   dev;
   int     flags;
};

struct socksysreq {
   int     args[7];
};
```

Figure 6-26: Header File: `<sys/types.h>`

```
#ifndef   FD_SETSIZE
#define   FD_SETSIZE   1024
#endif

#ifndef NBBY
#define NBBY 8
#endif

typedef   long     fd_mask;
#define   NFDBITS  (sizeof(fd_mask) * NBBY)
#ifndef   howmany
#define   howmany(x, y)  (((x)+((y)-1))/(y))
#endif

typedef   struct fd_set {
   fd_mask   fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

#define  FD_SET(n, p)   ((p)->fds_bits[(n)/NFDBITS] \
                     |= (1 << ((n) % NFDBITS)))
#define  FD_CLR(n, p)   ((p)->fds_bits[(n)/NFDBITS] \
                     &= ~(1 << ((n) % NFDBITS)))
#define  FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] \
                     &  (1 << ((n) % NFDBITS)))
#define  FD_ZERO(p)     memset((char *)(p), 0, sizeof(*(p)))
```

Figure 6-27: Header File: `<sys/un.h>`

```
struct  sockaddr_un {
   short   sun_family;
   char    sun_path[108];
};
```

ABI Library

NOTE The entire ABI Library section is new as of Version 1.2 of the Conformance Guide, but will not be marked with diffmarks.

Asynchronous Input and Output

The library `libabi.so.1` contains routines which perform asynchronous I/O operations. These are based on the asynchronous I/O routines documented in the POSIX 1003.1b (Real-Time) standard, but using callback functions for completion notification rather than real-time signals. This allows for more flexible operation and, in some implementations, increased performance. Since it is based on the POSIX standard, that standard should be consulted in concert with this one for full details.

Figure 6-28: `libabi` **Contents**

`aio_cancel*`	`aio_error*`	`aio_fsync*`
`aio_hold*`	`aio_read*`	`aio_return*`
`aio_suspend*`	`aio_write*`	`lio_listio*`

*Function is new to version 1.2

These routines have the equivalent behavior, except for the restrictions listed on page 6-34, to routines in the POSIX 1003.1b specification. The `aio_hold` routine does not have a POSIX equivalent.

Data Definitions

The header file `<aio.h>` includes definitions of the data structures and constants used by many of the asynchronous I/O function interfaces. It should also include the function prototypes.

Data structures and constants related to completion notification are defined in `<signal.h>`.

Defines for hard-coded operational limits are in `<limits.h>`.

Error codes for asynchronous I/O are in `<errno.h>`.

File `open` modes and `fcntl` options for asynchronous I/O are in `<fcntl.h>`.

Defines allowing querying for run-time limits are in `<unistd.h>`.

NOTE In the following figures, the C-style comment text is intended to be descriptive, but does not itself form part of the standard.

Figure 6-29: Header File: aio.h

```
/* for aio_cancel() return values */
#define AIO_CANCELED            1       /* all operations canceled */
#define AIO_NOTCANCELED         2       /* some ops could not be canceled */
#define AIO_ALLDONE             3       /* no ops could be canceled, all done */

/* for aiocb.lio_opcode */
#define LIO_NOP                 0       /* listio request with no data xfer */
#define LIO_READ                1       /* listio read request */
#define LIO_WRITE               2       /* listio write request */

/* for lio_listio mode flag */
#define LIO_WAIT                4       /* suspend until lio_listio complete */
#define LIO_NOWAIT              3       /* do not suspend or notify for lio_listio */

/* for aio_hold routine */
#define AIO_HOLD_CALLBACK       1
#define AIO_RELEASE_CALLBACK    2
#define AIO_ISHELD_CALLBACK     3

typedef struct aiocb {
   int                  aio_fildes;     /* file descriptor */
   volatile void        *aio_buf;       /* data buffer location */
   size_t               aio_nbytes;     /* number bytes of data */
   off_t                aio_offset;     /* file offset */
   int                  aio_reqprio;    /* request priority */
   struct sigevent      aio_sigevent;   /* notification method */
   int                  aio_lio_opcode; /* listio operation */
   ulong                aio_reserved[7]; /* reserved for internal use */
   ulong                aio_pad[6];
} aiocb_t;

/* prototypes */
int aio_read(aiocb_t *);
int aio_write(aiocb_t *);
int lio_listio(int, aiocb_t * const [], int, sigevent_t *);
int aio_error(const aiocb_t *);
int aio_return(aiocb_t *);
int aio_cancel(int, aiocb_t *);
int aio_suspend(const aiocb_t * const [], int, const struct timespec *);
int aio_fsync(int op, aiocb_t *);
int aio_hold(int);
```

Figure 6-30: Header File: signal.h

```
typedef union {
   int                  nisigno;                /* signal number */
   void                 (*nifunc)(union sigval); /* callback data */
} notifyinfo_t;

typedef union sigval {
   int                  sival_int;              /* integer notify value */
   void                 *sival_ptr;             /* pointer notify value */
} sigval_t;

typedef struct sigevent {
    int                 sigev_notify;           /* notification type */
    notifyinfo_t        sigev_notifyinfo;       /* notification info */
    sigval_t            sigev_value;            /* notification handler argument */
    ulong               sigev_reserved[13];     /* reserved for internal use */
    ulong               sigev_pad[6];           /* reserved for future use */
} sigevent_t;

#define sigev_func sigev_notifyinfo.nifunc
#define sigev_signo sigev_notifyinfo.nisigno

/* values of sigev_notify */
#define SIGEV_NONE              128             /* no async notification */
#define SIGEV_SIGNAL            129             /* queued signal notification
                                                   NOT SUPPORTED for MIPS ABI */
#define SIGEV_CALLBACK          130             /* callback notification */
```

Figure 6-31: Header File: limits.h

```
#define _POSIX_AIO_LISTIO_MAX    2
#define _POSIX_AIO_MAX           1

#define _ABI_AIO_XFER_MAX        (128*1024)      /* minimum max value */
```

Figure 6-32: Header File: errno.h

```
#define EINPROGRESS      150
#define ECANCELED        158
```

Figure 6-33: Header File: fcntl.h

```
#define O_DSYNC         0x20
#define O_RSYNC         0x40
```

Figure 6-34: Header File: unistd.h

```
#define _SC_ASYNCHRONOUS_IO      64              /* is POSIX AIO supported? */
#define _SC_ABI_ASYNCHRONOUS_IO  65              /* is ABI AIO supported? */
#define _SC_AIO_LISTIO_MAX       66              /* max listio operation */
#define _SC_AIO_MAX              67              /* max AIO */
#define _PC_ASYNC_IO             64              /* can this file do POSIX AIO? */
#define _PC_ABI_ASYNC_IO         65              /* can this file do ABI AIO? */
#define _PC_ABI_AIO_XFER_MAX     66              /* biggest ABI AIO xfer for this file */
```

NOTE

`_SC_ASYNCHRONOUS_IO` and `_PC_ASYNC_IO` are included for completeness, for checking for strict POSIX conformance. An ABI-conforming application would not query for POSIX, but rather for the ABI values.

Function Interfaces

The operation and error returns from all functions except `aio_hold` are those of the corresponding POSIX 1003.1b definition with the substitution of callback functions for notification by real-time signals. In addition to the specified error returns, the functions which correspond to POSIX 1003.1b functions can also set *errno* to the following value:

`EBADF`	Asynchronous I/O is not supported on the given file descriptor.

`int aio_read(aiocb_t *aiocbp)`
: asynchronous read

`int aio_write(aiocb_t *aiocbp)`
: asynchronous write

`int lio_listio(int mode, aiocb_t * const list[], int nent, sigevent_t *sig)`
: list directed I/O

`int aio_error(const aiocb_t *aiocbp)`
: retrieve error status of asynchronous I/O operation

`int aio_return(aiocb_t *aiocbp)`
: retrieve return status of asynchronous I/O operation

`int aio_cancel(int fildes, aiocb_t *aiocbp)`
cancel asynchronous I/O request

`int aio_suspend(const aiocb_t * const list[], int nent, const struct timespec *timeout)`
wait for asynchronous I/O request

`int aio_fsync(int op, aiocb_t *iocbp)`
asynchronous file synchronization

`int aio_hold(int should_hold)`
Defer or resume reception of callback invocations. If the *should_hold* parameter is `AIO_HOLD_CALLBACK`, callbacks will be deferred; if `AIO_RELEASE_CALLBACK`, callback invocation will resume. `aio_hold` counts calls, and only when the count of `AIO_HOLD_CALLBACK` calls reaches zero via a corresponding number of `AIO_RELEASE_CALLBACK` calls will callbacks be released. Upon successful completion, the state prior to the call is returned. A return value of `1` indicates there were one or more hold calls, a value of `0` indicates callbacks were not being held. Otherwise, a value of `–1` is returned and *errno* is set to indicate the error. `aio_hold` can also be called with a parameter of `AIO_ISHELD_CALLBACK` which will query the current state; it returns `1` if callbacks are currently held, `0` otherwise. When a callback routine is called, callbacks are automatically held.

`aio_hold` will fail if one or more of the following are true:

`EINVAL`	The value of *should_hold* is not one of `AIO_ISHELD_CALLBACK`, `AIO_HOLD_CALLBACK` or `AIO_RELEASE_CALLBACK`.
`ENOSYS`	Asynchronous I/O is not supported on the current system.
`EIO`	It was not possible to perform this operation due to an error.

Implementation

The element *sigev_notify* in the *sigevent* structure may have the values `SIGEV_NONE` or `SIGEV_CALLBACK`. These values, as well as `SIGEV_SIGNAL` (which is not used but reserved) are chosen to have values so that an implementation can easily distinguish calls to the ABI implementation of async I/O from similar co-existing implementations. In particular, these values are chosen to be larger than `MAXSIG`.

The element *nisigno* in the *notifyinfo* union is unused in this version, but is provided for possible future changes. If the *sigev_notify* element of the *sigevent* structure is set to `SIGEV_CALLBACK`, *sigev_notifyinfo.nifunc* (which can also be referred to as *sigev_func*) indicates the function to be invoked when the operation completes. That function will be called with a single argument, which is the value in *sigev_value*, and will use the current stack indicated by the `$sp` register.

The execution context in which callback functions are invoked is undefined, however the callback has access to all global data and functions.

All operations require the *aio_reqprio* and *pad* fields to be set to zero.

In keeping with the POSIX 1003.1b standard, limits are generally not hardcoded. Instead, applications should use `sysconf` to query for system-wide limits, and `pathconf` for per-file limits.

Unlike in the POSIX standard, no extensions to the data structures are allowed. The implementation may use the reserved fields for internal purposes.

Restrictions

The following restrictions must be observed by applications using this facility:

- The reserved fields are for internal use by various implementations and should not be used by applications.
- The behavior in the presence of the definition of `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` is undefined. Among other things, this means that the behavior when *aio_reqprio* is non-zero is undefined.
- There is a maximum number of bytes that may be transferred in a single request. An application may either use the minimum maximum (`_ABI_AIO_XFER_MAX`) or query `pathconf` using the `_PC_ABI_AIO_XFER_MAX` parameter (preferred) to determine the limit for the file in question.
- If asynchronous I/O is not supported on a given file descriptor, i.e. `pathconf(_PC_ABI_ASYNC_IO)` would return a value other than `1`, the asynchronous I/O functions will return an error with *errno* set to `EBADF`.
- Use of the `O_SYNC` flag with `open` will ensure that completion notification will occur only after the data has been committed to be written to persistent storage.
- Behavior due to modification of the contents of an *aiocb* structure between the time an operation is queued and the operation completes is undefined. The application should use `aio_error` to check for completion if the status is unknown.
- The behavior when *sigevent.sigev_notify* is set to `SIGEV_SIGNAL` is undefined.
- The application must return from a callback function before the next callback function is invoked. If an application exits or does a `longjmp` out of a callback, the future use of any asynchronous I/O request is undefined.
- Some devices may not support all I/O sizes and offsets. Operations on such devices will return `–1` and set *errno* accordingly to indicate that an unsupported operation has occurred.
- The number of outstanding callbacks may be limited by a system resource limit, possibly a tunable variable. If the limit is exceed, calls will begin to fail and set *errno* to `EAGAIN` until resources free up.

NOTE

Application vendors are cautioned that these interfaces are present only in systems which comply with Version 1.2 and above of this document. These interfaces are new to the ABI and may not be immediately available on all platforms.

Operating system implementors are advised that parallelized `lio_listio` implementations, in which many I/O operations are in progress at a given time are strongly desired by the DBMS community. Simply serially processing the `lio_listio` call may not be of much benefit to the DBMS community.

If the implementation chooses to use threads, then the system-supplied libraries must be thread-safe.

X Window System Library

Graphical MIPS ABI-conforming applications may be constructed using the X Window System, Version 11, Release 5 (X11R5). The generic ABI specifies X11R3 but indicates an intent to track upward compatible future releases of the X Window System. Prior versions of this document relied on the generic ABI specification of X11R4 and did not have MIPS ABI extensions in this area. This library is required on all ABI-conforming systems.

The basic X11 support library, `libX`, shall be supported as a DSO, `libX11.so.2`, by ABI-conforming systems. (`libX11.so.1` is reserved for implementations that may have had X11R4 versions.) The X Toolkit Intrinsics library, `libXt`, is supplied as a static archive in the ABI build environment.

The following functions reside in `libX` and must be provided on all ABI-conforming systems.

Figure 6-35: `libX` **Contents**

XActivateScreenSaver	XAddExtension	XAddHost
XAddHosts	XAddPixel	XAddToExtensionList
XAddToSaveSet	XAllocClassHint	XAllocColor
XAllocColorCells	XAllocColorPlanes	XAllocIconSize
XAllocNamedColor	XAllocSizeHints	XAllocStandardColormap
XAllocWMHints	XAllowEvents	XAllPlanes
XAutoRepeatOff	XAutoRepeatOn	XBaseFontNameListOfFontSet
XBell	XBitmapBitOrder	XBitmapPad
XBitmapUnit	XBlackPixel	XBlackPixelOfScreen
XCellsOfScreen	XChangeActivePointerGrab	XChangeGC
XChangeKeyboardControl	XChangeKeyboardMapping	XChangePointerControl
XChangeProperty	XChangeSaveSet	XChangeWindowAttributes
XCheckIfEvent	XCheckMaskEvent	XCheckTypedEvent
XCheckTypedWindowEvent	XCheckWindowEvent	XCirculateSubwindows
XCirculateSubwindowsDown	XCirculateSubwindowsUp	XClearArea
XClearWindow	XClipBox	XCloseDisplay
XCloseIM	XcmsAddColorSpace	XcmsAddFunctionSet
XcmsAllocColor	XcmsAllocNamedColor	XcmsCCCOfColormap
XcmsCIELabClipab	XcmsCIELabClipL	XcmsCIELabClipLab
XcmsCIELabQueryMaxC	XcmsCIELabQueryMaxL	XcmsCIELabQueryMaxLC
XcmsCIELabQueryMinL	XcmsCIELabToCIEXYZ	XcmsCIELabWhiteShiftColors
XcmsCIELuvClipL	XcmsCIELuvClipLuv	XcmsCIELuvClipuv
XcmsCIELuvQueryMaxC	XcmsCIELuvQueryMaxL	XcmsCIELuvQueryMaxLC
XcmsCIELuvQueryMinL	XcmsCIELuvToCIEuvY	XcmsCIELuvWhiteShiftColors
XcmsCIEuvYToCIELuv	XcmsCIEuvYToCIEXYZ	XcmsCIEuvYToTekHVC
XcmsCIExyYToCIEXYZ	XcmsCIEXYZToCIELab	XcmsCIEXYZToCIEuvY
XcmsCIEXYZToCIExyY	XcmsCIEXYZToRGBi	XcmsClientWhitePointOfCCC
XcmsConvertColors	XcmsCreateCCC	XcmsDefaultCCC
XcmsDisplayOfCCC	XcmsFormatOfPrefix	XcmsFreeCCC
XcmsLookupColor	XcmsPrefixOfFormat	XcmsQueryBlack
XcmsQueryBlue	XcmsQueryColor	XcmsQueryColors
XcmsQueryGreen	XcmsQueryRed	XcmsQueryWhite
XcmsRGBiToCIEXYZ	XcmsRGBiToRGB	XcmsRGBToRGBi
XcmsScreenNumberOfCCC	XcmsScreenWhitePointOfCCC	XcmsSetCCCOfColormap
XcmsSetCompressionProc	XcmsSetWhiteAdjustProc	XcmsSetWhitePoint
XcmsStoreColor	XcmsStoreColors	XcmsTekHVCClipC
XcmsTekHVCClipV	XcmsTekHVCClipVC	XcmsTekHVCQueryMaxC
XcmsTekHVCQueryMaxV	XcmsTekHVCQueryMaxVC	XcmsTekHVCQueryMaxVSamples

Figure 6-35: `libX` **Contents** (continued)

XcmsTekHVCQueryMinV	XcmsTekHVCToCIEuvY	XcmsTekHVCWhiteShiftColors
XcmsVisualOfCCC	XConfigureWindow	XConnectionNumber
XContextDependentDrawing	XConvertSelection	XCopyArea
XCopyColormapAndFree	XCopyGC	XCopyPlane
XCreateBitmapFromData	XCreateColormap	XCreateFontCursor
XCreateFontSet	XCreateGC	XCreateGlyphCursor
XCreateIC	XCreateImage	XCreatePixmap
XCreatePixmapCursor	XCreatePixmapFromBitmapData	XCreateRegion
XCreateSimpleWindow	XCreateWindow	XDefaultColormap
XDefaultColormapOfScreen	XDefaultDepth	XDefaultDepthOfScreen
XDefaultGC	XDefaultGCOfScreen	XDefaultRootWindow
XDefaultScreen	XDefaultScreenOfDisplay	XDefaultString
XDefaultVisual	XDefaultVisualOfScreen	XDefineCursor
XDeleteContext	XDeleteModifiermapEntry	XDeleteProperty
XDestroyIC	XDestroyImage	XDestroyRegion
XDestroySubwindows	XDestroyWindow	XDisableAccessControl
XDisplayCells	XDisplayHeight	XDisplayHeightMM
XDisplayKeycodes	XDisplayMotionBufferSize	XDisplayName
XDisplayOfIM	XDisplayOfScreen	XDisplayPlanes
XDisplayString	XDisplayWidth	XDisplayWidthMM
XDoesBackingStore	XDoesSaveUnders	XDrawArc
XDrawArcs	XDrawImageString	XDrawImageString16
XDrawLine	XDrawLines	XDrawPoint
XDrawPoints	XDrawRectangle	XDrawRectangles
XDrawSegments	XDrawString	XDrawString16
XDrawText	XDrawText16	XEHeadOfExtensionList
XEmptyRegion	XEnableAccessControl	XEqualRegion
XESetCloseDisplay	XESetCopyGC	XESetCreateFont
XESetCreateGC	XESetError	XESetErrorString
XESetEventToWire	XESetFlushGC	XESetFreeFont
XESetFreeGC	XESetPrintErrorValues	XESetWireToError
XESetWireToEvent	XEventMaskOfScreen	XEventsQueued
XExtentsOfFontSet	XFetchBuffer	XFetchBytes
XFetchName	XFillArc	XFillArcs
XFillPolygon	XFillRectangle	XFillRectangles
XFilterEvent	XFindContext	XFindOnExtensionList
XFlush	XFlushGC	XFontsOfFontSet
XForceScreenSaver	XFree	XFreeColormap
XFreeColors	XFreeCursor	XFreeExtensionList
XFreeFont	XFreeFontInfo	XFreeFontNames
XFreeFontPath	XFreeFontSet	XFreeGC
XFreeModifiermap	XFreePixmap	XFreeStringList
XGContextFromGC	XGeometry	XGetAtomName
XGetClassHint	XGetCommand	XGetDefault
XGetErrorDatabaseText	XGetErrorText	XGetFontPath
XGetFontProperty	XGetGCValues	XGetGeometry
XGetIconName	XGetIconSizes	XGetICValues
XGetImage	XGetIMValues	XGetInputFocus
XGetKeyboardControl	XGetKeyboardMapping	XGetModifierMapping

Figure 6-35: `libX` **Contents** (continued)

XGetMotionEvents	XGetNormalHints	XGetPixel
XGetPointerControl	XGetPointerMapping	XGetRGBColormaps
XGetScreenSaver	XGetSelectionOwner	XGetSizeHints
XGetStandardColormap	XGetSubImage	XGetTextProperty
XGetTransientForHint	XGetVisualInfo	XGetWindowAttributes
XGetWindowProperty	XGetWMClientMachine	XGetWMColormapWindows
XGetWMHints	XGetWMIconName	XGetWMName
XGetWMNormalHints	XGetWMProtocols	XGetWMSizeHints
XGetZoomHints	XGrabButton	XGrabKey
XGrabKeyboard	XGrabPointer	XGrabServer
XHeightMMOfScreen	XHeightOfScreen	XIconifyWindow
XIfEvent	XImageByteOrder	XIMOfIC
XInitExtension	XInsertModifiermapEntry	XInstallColormap
XInternAtom	XIntersectRegion	XKeycodeToKeysym
XKeysymToKeycode	XKeysymToString	XKillClient
XLastKnownRequestProcessed	XListDepths	XListExtensions
XListFonts	XListFontsWithInfo	XListHosts
XListInstalledColormaps	XListPixmapFormats	XListProperties
XLoadFont	XLoadQueryFont	XLocaleOfFontSet
XLocaleOfIM	XLookupColor	XLookupKeysym
XLookupString	XLowerWindow	XMapRaised
XMapSubwindows	XMapWindow	XMaskEvent
XMatchVisualInfo	XMaxCmapsOfScreen	XMaxRequestSize
XmbDrawImageString	XmbDrawString	XmbDrawText
XmbLookupString	XmbResetIC	XmbSetWMProperties
XmbTextEscapement	XmbTextExtents	XmbTextListToTextProperty
XmbTextPerCharExtents	XmbTextPropertyToTextList	XMinCmapsOfScreen
XMoveResizeWindow	XMoveWindow	XNewModifiermap
XNextEvent	XNextRequest	XNoOp
XOffsetRegion	XOpenDisplay	XOpenIM
XParseColor	XParseGeometry	XPeekEvent
XPeekIfEvent	XPending	Xpermalloc
XPlanesOfScreen	XPointInRegion	XPolygonRegion
XProtocolRevision	XProtocolVersion	XPutBackEvent
XPutImage	XPutPixel	XQLength
XQueryBestCursor	XQueryBestSize	XQueryBestStipple
XQueryBestTile	XQueryColor	XQueryColors
XQueryExtension	XQueryFont	XQueryKeymap
XQueryPointer	XQueryTextExtents	XQueryTextExtents16
XQueryTree	XRaiseWindow	XReadBitmapFile
XRebindKeysym	XRecolorCursor	XReconfigureWMWindow
XRectInRegion	XRefreshKeyboardMapping	XRemoveFromSaveSet
XRemoveHost	XRemoveHosts	XReparentWindow
XResetScreenSaver	XResizeWindow	XResourceManagerString
XRestackWindows	XrmCombineDatabase	XrmCombineFileDatabase
XrmDestroyDatabase	XrmEnumerateDatabase	XrmGetDatabase
XrmGetFileDatabase	XrmGetResource	XrmGetStringDatabase
XrmInitialize	XrmLocaleOfDatabase	XrmMergeDatabases
XrmParseCommand	XrmPermStringToQuark	XrmPutFileDatabase

Figure 6-35: `libX` **Contents** (continued)

XrmPutLineResource	XrmPutResource	XrmPutStringResource
XrmQGetResource	XrmQGetSearchList	XrmQGetSearchResource
XrmQPutResource	XrmQPutStringResource	XrmQuarkToString
XrmSetDatabase	XrmStringToBindingQuarkList	XrmStringToQuark
XrmStringToQuarkList	XrmUniqueQuark	XRootWindow
XRootWindowOfScreen	XRotateBuffers	XRotateWindowProperties
XSaveContext	XScreenCount	XScreenNumberOfScreen
XScreenOfDisplay	XScreenResourceString	XSelectInput
XSendEvent	XServerVendor	XSetAccessControl
XSetAfterFunction	XSetArcMode	XSetBackground
XSetClassHint	XSetClipMask	XSetClipOrigin
XSetClipRectangles	XSetCloseDownMode	XSetCommand
XSetDashes	XSetErrorHandler	XSetFillRule
XSetFillStyle	XSetFont	XSetFontPath
XSetForeground	XSetFunction	XSetGraphicsExposures
XSetICFocus	XSetIconName	XSetIconSizes
XSetICValues	XSetInputFocus	XSetIOErrorHandler
XSetLineAttributes	XSetLocaleModifiers	XSetModifierMapping
XSetNormalHints	XSetPlaneMask	XSetPointerMapping
XSetRegion	XSetRGBColormaps	XSetScreenSaver
XSetSelectionOwner	XSetSizeHints	XSetStandardColormap
XSetStandardProperties	XSetState	XSetStipple
XSetSubwindowMode	XSetTextProperty	XSetTile
XSetTransientForHint	XSetTSOrigin	XSetWindowBackground
XSetWindowBackgroundPixmap	XSetWindowBorder	XSetWindowBorderPixmap
XSetWindowBorderWidth	XSetWindowColormap	XSetWMClientMachine
XSetWMColormapWindows	XSetWMHints	XSetWMIconName
XSetWMName	XSetWMNormalHints	XSetWMProperties
XSetWMProtocols	XSetWMSizeHints	XSetZoomHints
XShrinkRegion	XStoreBuffer	XStoreBytes
XStoreColor	XStoreColors	XStoreName
XStoreNamedColor	XStringListToTextProperty	XStringToKeysym
XSubImage	XSubtractRegion	XSupportsLocale
XSync	XSynchronize	XTextExtents
XTextExtents16	XTextPropertyToStringList	XTextWidth
XTextWidth16	XTranslateCoordinates	XUndefineCursor
XUngrabButton	XUngrabKey	XUngrabKeyboard
XUngrabPointer	XUngrabServer	XUninstallColormap
XUnionRectWithRegion	XUnionRegion	XUnloadFont
XUnmapSubwindows	XUnmapWindow	XUnsetICFocus
XVaCreateNestedList	XVendorRelease	XVisualIDFromVisual
XWarpPointer	XwcDrawImageString	XwcDrawString
XwcDrawText	XwcFreeStringList	XwcLookupString
XwcResetIC	XwcTextEscapement	XwcTextExtents
XwcTextListToTextProperty	XwcTextPerCharExtents	XwcTextPropertyToTextList
XWhitePixel	XWhitePixelOfScreen	XWidthMMOfScreen
XWidthOfScreen	XWindowEvent	XWithdrawWindow
XWMGeometry	XWriteBitmapFile	XXorRegion
_XAllocScratch	_XEatData	_XFlush

Figure 6-35: `libX` **Contents** (continued)

_XFlushGCCache	_XRead	_XReadEvents
_XReadPad	_XReply	_XSend
_XSetLastRequestRead		

The `libX` library requires that some global external data symbols be defined for its' routines to work properly. All the data symbols listed in the table below must be provided by the `libX` library.

Figure 6-36: `libX` **Contents, Global External Data Symbols**

XcmsCIELabColorSpace	XcmsCIELuvColorSpace
XcmsCIEuvYColorSpace	XcmsCIExyYColorSpace
XcmsCIEXYZColorSpace	XcmsLinearRGBFunctionSet
XcmsRGBColorSpace	XcmsRGBiColorSpace
XcmsTekHVCColorSpace	XcmsUNDEFINEDColorSpace

An ABI-conforming system shall provide the following include files whose content is defined by the X11R5 Reference Manual.

Figure 6-37: X11R5 Include Files

X11/cursorfont.h	X11/keysymdef.h	X11/X.h
X11/Xatom.h	X11/Xcms.h	X11/Xlib.h
X11/Xproto.h	X11/Xresource.h	X11/Xutil.h

Figure 6-38: X11R5 X Toolkit Intrinsics Include Files

X11/Intrinsic.h	X11/IntrinsicP.h	X11/Shell.h
X11/ShellP.h	X11/StringDefs.h	X11/Vendor.h
X11/VendorP.h		

Math Library

Frequently, applications must rely on groups of object files not required to be present on an ABI conforming implementation. These may be provided in static archives provided with the development environment. If each member of the archive is itself ABI conforming, then an ABI conforming application may statically link members from this archive and still be ABI conforming. If extensions to an archive are not ABI conforming, then an ABI conforming application may not include that extension in an executable.

All development environments for ABI applications shall contain ABI conforming versions of `libm.a`. The MIPS processor specific ABI shall define the path to the directory that contains these libraries. The following are entry-points that must be defined for each respective library, as defined in the *System V Interface Definition, Third Edition*.

Figure 6-39: Required `libm` Functions

acos	acosh	asin	asinh	atan*	atan2
atanh	cbrt	ceil	cos	cosh	erf
erfc	exp	fabs	floor	fmod	gamma
hypot*	j0	j1	jn	lgamma	log
log10	pow	remainder	sin	sinh	sqrt
tan	tanh	y0	y1	yn	

*Function was erroneously omitted from earlier editions

OSF/Motif Library

An ABI-conforming system shall support Release 1.2 of the OSF/Motif software as a static library in the ABI build environment, `libXm.a`. The first ABI reference platform (corresponding to version 1.0 of this specification) supported OSF/Motif 1.1 as a static library.

An ABI-conforming system shall provide the following include files whose content is defined by the *OSF/Motif Programmer's Reference Manual*.

Figure 6-40: OSF/Motif 1.2 Include Files

`Xm/ArrowB.h`	`Xm/ArrowBG.h`	`Xm/BulletinB.h`
`Xm/CascadeB.h`	`Xm/CascadeBG.h`	`Xm/Command.h`
`Xm/CutPaste.h`	`Xm/DialogS.h`	`Xm/DrawingA.h`
`Xm/DrawnB.h`	`Xm/FileSB.h`	`Xm/Form.h`
`Xm/Frame.h`	`Xm/Label.h`	`Xm/LabelG.h`
`Xm/List.h`	`Xm/MainW.h`	`Xm/MenuShell.h`
`Xm/MessageB.h`	`Xm/PanedW.h`	`Xm/Protocols.h`
`Xm/PushB.h`	`Xm/PushBG.h`	`Xm/RowColumn.h`
`Xm/Scale.h`	`Xm/ScrollBar.h`	`Xm/ScrolledW.h`
`Xm/SelectioB.h`	`Xm/SeparatoG.h`	`Xm/Separator.h`
`Xm/Text.h`	`Xm/ToggleB.h`	`Xm/ToggleBG.h`
`Xm/Xm.h`	`Xm/XmP.h`	

General-Purpose Library

These functions constitute a general-purpose library, `libgen`. This library is not implemented as a shared object. It is required to link OSF/Motif applications.

Figure 6-41: `libgen` **Contents**

basename	bgets	bufsplit	copylist
dirname	eaccess	gmatch	isencrypt
mkdirp	p2open	p2close	pathfind
regcmp	regex	rmdirp	strcadd
strccpy	streadd	strecpy	strfind
strrspn	strtrns		
compile*	step*	advance*	

* Listed as Level 2 in the SVID

These functions are declared in `<libgen.h>`.

```
char *basename(char *);
char *bgets(char *, size_t, FILE *, char *);
size_t bufsplit(char *, size_t, const char *);
char *copylist(const char *, off_t *);
char *dirname(char *);
int eaccess(const char *, int);
int gmatch(const char *, const char *);
int isencrypt(const char *, size_t);
int mkdirp(const char *, mode_t);
int p2open(const char *, FILE *[2]);
int p2close(FILE *[2]);
char *pathfind(const char *, const char *, const char *);
char *regcmp(const char *, ...);
char *regex(const char *, const char *, ...);
int rmdirp(char *, char *);
char *strccpy(char *, const char *);
char *strcadd(char *, const char *);
char *strecpy(char *, const char *, const char *);
char *streadd(char *, const char *, const char *);
int strfind(const char *, const char *);
char *strrspn(const char *, const char *);
char *strtrns(const char *, const char *, const char *, char *);
```

These functions are declared in `<regexpr.h>`, and operate as described in the *System V Interface Definition, Third Edition (SVID 3)*. Note that the description in the SVID refers to a slightly different implementation, and refereneces a different header file, `<regexp.h>`, not the `libgen` implementation. In any case, these routines are marked as level 2 and should be avoided, as they are not required to be present on an ABI-conforming system.

```
char *compile(const char *, char *, char *);
int step(const char *, const char *);
int advance(const char *, const char *);
```

BSD Emulation Library

`libucb.a` provides many of the popular interfaces from BSD UNIX. It is intended to be used as a migration path for ISVs moving applications from BSD UNIX to SVR4. Over time all ISVs are encouraged to move to the SVR4 ABI interfaces since they may not be supported in the future. It is also important to note that use of `libucb.a` pulls in all of its BSD interfaces. This can result in subtle problems for applications which mix BSD and SVR4 interfaces, for example using the BSD `printf` and SVR4 `readdir`. In the previous case, both the `printf` and `readdir` calls would use the BSD version. This class of problem will be exposed at runtime rather than compile time.

Figure 6-42: `libucb` **Contents**

alloca	alphasort	bcmp
bcopy	bzero	dbm_close
dbm_delete	dbm_do_nextkey	dbm_fetch
dbm_firsthash	dbm_firstkey	dbm_forder
dbm_nextkey	dbm_open	dbm_store
dc	endusershell	fopen
fp_accrued_exceptions	fp_class_d	fp_class_f
fp_direction	fp_precision	fprintf
ftime	getdtablesize	gethostid
gethostname	getpagesize	getrusage
getusershell	getwd	ieee_handlers
index	initstate	isnan
killpg	longjmp	mctl
mkstemp	printf	psignal
rand	random	re_comp
re_exec	readdir	rindex
scandir	sethostname	setjmp
setstate	setusershell	sigblock
sigfpe	siginterrupt	signal
sigpause	sigsetmask	sigstack
sigvec	sleep	sprintf
srand	srandom	strcasecmp
strncasecmp	sys_siglist	times
timezone	ualarm	usleep
utimes	vfprintf	vprintf
vsprintf	wait3	

WARNING The BSD Emulation Library is at level 2, and is likely to be removed in the next edition of this specification.

7 FORMATS AND PROTOCOLS

ABI Version Identification

An ABI-conforming system which conforms to revision 1.2 or later of the ABI Conformance Guide will have an ASCII file which identifies the highest revision level to which the system conforms.

The version file is named `/etc/mipsabiversion`.

The file contains a newline-terminated string of the form *major.minor*. Only the first line should be processed; all subsequent lines should be ignored. Sub-versions (e.g., the .1 in 1.0.1) are not reflected in the version file as they only reflect clarifications to the wording in a baseline spec.

If the file is not found, the system should only be assumed to be conformant to a Conformance Guide revision prior to 1.2 - it is not possible to distinguish between revision 1.0 and revision 1.1.

8 SYSTEM COMMANDS

System Commands

In addition to those commands listed in the generic ABI and the Processor Specific ABI, the following commands are available to application programs running on ABI-conforming systems:

- `-D` option to `cpio`
- `compress`
- `mount`
- `mt`
- `tar`
- `uncompress`
- `umount`
- `zcat`

With the exception of the `-D` option to `cpio`, and `mt`, `mount` and `umount`, these commands are as described in the *System V Interface Definition, Third Edition (SVID 3)*.

The `-D` option to cpio

The `cpio` command must support the `-D` option. `-D` is used by the SVR4 packaging tools, and has been found to be necessary for installation of third party software which resides on multi-file tapes. `-D` disables tape read ahead and automatic volume switching.

The mt Command

The `mt` command is a magnetic tape manipulation program, with the following syntax:

`mt -f` *tapename command* `[` *count* `]`

`mt` is used to give commands to a magnetic tape drive. Note that *tapename* must reference a raw (not block) tape device. By default, `mt` performs the requested operation once. Operations may be performed multiple times by specifying *count*.

Valid choices for *command* are as follows:

`fsf`	Forward space count files.
`rewind`	Rewind the tape (count is ignored).

NOTE A rewind media command for a partitioned 4mm DDS tape will take you to the first block of the current partition, not the physical beginning of media. The specification of commands to deal with partitioned DDS media is outside the current scope of the ABI.

`mt` returns a 0 exit status upon successful completion, 1 if the command was unrecognized, and 2 if an operation failed.

NOTE Differences are known to exist among various vendor platforms as to exactly where a tape is positioned after reading a file on a multi-file tape. These are difficult to reconcile due to compatibility requirements with prior OS releases by each vendor. Applications that use multi-file tapes, especially using them for installation should consider using `dd` to actually extract the information, and the undocumented `-D` option for `cpio` (discussed above) which inhibits read ahead. This is the mechanism used by the SVR4 packaging tools. Vendors have generally made `dd` properly position the tape in a consistent way.

The mount and umount Commands

The `mount` and `unmount` commands specify how to mount and unmount file systems. The specification below extends the definition in the *SVID 3* to support the mounting and unmounting of CD-ROM based file systems. This definition is derived from the OCMP and SVR4.2 specifications.

If *FSType* is `cdfs`, the *special* argument must be `/dev/abi/cd_iso`. Note that *FSType* and/or *special* can in some circumstances be determined from `/etc/vfstab`.

The `cdfs`-specific options to `mount` include:

`ro` — Mounts the resource read-only. This option or the `-r` option are required.

`susp|nosusp` — Enables/disables the processing of all System Use Sharing Protocol (SUSP) compliant extensions to the ISO-9660/High Sierra specification. The SUSP defines a mechanism for which the System Use Area of a Directory Record may be shared by multiple independent organizations, for example, Rock Ridge Interchange Protocol (RRIP). The `nosusp` flag prevents the processing of all SUSP-compliant extensions, even if recorded on the media.

`rrip|norrip` — Enables/disables the processing of all the Rock Ridge Interchange Protocol (RRIP) extensions to the ISO-9660/High Sierra specification. The RRIP is a SUSP-compliant set of extensions that provide POSIX file semantics within the context of an ISO-9660/High Sierra file-system. The `rrip` flag requires that the `susp` flag also be set. The `norrip` flag prevents the processing of all RRIP extensions even if recorded on the media.

`nmconv=`*a* — Sets the file name conversion mode for the ISO-9660/High Sierra file and directory names recorded on the CD-ROM. The `nmconv` option lets you specify how the file and directory names are seen by the user on the host system. Non-ISO-9660 and non-High Sierra names are not converted.

a is some combination of the following:

- `c` — Names are not converted; the names used are the ones recorded on the CD-ROM.
- `l` — Converts upper case letters to lower case.
- `m` — Suppresses the version number and the separator (;).

9 EXECUTION ENVIRONMENT

Execution Environment

NOTE

There are no extensions to execution environment in this volume.

10 WINDOWING AND TERMINAL INTERFACES

Windowing and Terminal Interfaces

The windowing system for the MIPS ABI is X11R5. ABI conformant applications should use the DSO library `libX11.so.2`. The X Toolkit Library, libXt, is supplied as a static archive `libXt.a` in the IRIX Reference Platform ABI Build environment. Due to inadequacies of the available test tools, it is not feasible to ensure that each vendor's implementation of libXt as a DSO is functionally equivalent. Also supplied as a static archive in the ABI build environment is the OSF/Motif library, `libXm.a`, constructed from the OSF/Motif Release 1.2.3 baseline. In the future, these libraries may become DSOs if adequate tests can be developed to ensure compatibility.

The ABI does not require that the system support an X Server, allowing for the possibility of server systems without local graphics capabilities.

NOTE The Window System is an optional component in the generic ABI. This chapter alters the concept so that libraries are manadatory and only the server is optional for conforming systems.

11 DEVELOPMENT ENVIRONMENT

Development Environment

> **NOTE** The facilities and interfaces described in this section are optional components of the System V Application Binary Interface.

Any system may be used to provide a development environment for ABI conforming applications. This chapter describes the commands, options, libraries, and path mechanisms necessary to produce an ABI conforming application. This development environment need not be hosted on an ABI conforming implementation.

Software Packaging Tools

A development environment for ABI applications shall include each of the following commands as defined in the ''AS_CMD'' section of the *System V Interface Definition, Third Edition (SVID 3)*.

- `pkgproto`
- `pkgtrans`
- `pkgmk`

The `pkgtrans` command shall generate output compliant with chapter 2 of the generic ABI.

A FUTURE DIRECTIONS

Future Directions

NOTE: This section is not part of the specification. It is provided to give an insight into development that is being considered for inclusion into a future version of this specification. It is not marked with diffmarks.

WARNING: The MIPS ABI Group offers no guarantees that any particular feature listed here will ever be part of the specification, or that any particular feature will take the form described here.

The application binary interface standards will continue to evolve over time to accomodate new technology. Future evolution will increase the capabilities defined in the ABI specifications and will expand the standard development environment provided for application writers and independent software vendors. This section describes issues the MIPS ABI Group is considering for inclusion in future revisions of this specification.

C++

Support for C++ application development and linking provided by means of a Compiler Environment concept. This Compiler Environment concept provides a complete package which itself is an ABI product. A full ABI for C++ where different C++ compilers can contribute objects to a binary is most desirable but is waiting for completion of the ISO C++ standardization effort.

Large File Access

A set of interfaces that allows access to files larger than can be described by 32-bit quantities and used on filesystems which can support them.

`long long`

A 64-bit quantity described to the C compilation system as `long long` as an integral type. This is primarily for use by the Large File interfaces.

X11R6

X11R6 (Release 6) as the supported version of the X Window System.

Network Installation

A standard mechanism for installing software over a network.

Evolution of Base Standards

The Version 1 releases of this specification were based on the first editions of the System V ABI and the MIPS Processor Supplement. The new base editions for Version 2 releases would be the 3rd editions of the gABI and psABI under this proposal.

Generation Platforms Support

Introduce a concept of *Generation Platforms* allowing ABI applications to be developed on machines besides the Reference Platform.

Symmetric Multiprocessing

Features to enable application programs to take advantage of SMP platforms. These include a system call to perform processor reservations and obtain status, and possibly support for the POSIX Threads programming model.

XPG 4 Unified Unix Interfaces

The XPG4.2 interfaces, formerly known as Spec 1170.

Optional Feature Interrogation

A mechanism that allows interrogation of a particular system to see whether it supports a particular feature. This capability allows both features designated as optional and early adoption of features (i.e., a platform may support a particular feature from a later specification without being conformant to the entire specification).

Expanded Dynamic Linking Control

Options to the compilation environment which will allow finer control over dynamic linking, including symbol visibility, control over run-time search paths, etc.

Portable stack backtrace

Specification of a mechanism for a stack backtrace of a program to be obtained without resorting to stepping through individually disassembled instructions.

Long-Term Projects

A number of projects which may appear in later editions of this specification, but certainly won't be in the next version.

Symbolic Debugging

Currently, full symbolic debugging is not possible except on the generation platform. Full support by means of specifying the */proc* interface and likely DWARF 2.0.

Line Printer Spooling

Interfaces for programmatic access to the printer spooling system.

OpenGL™

Specification of OpenGL as an optional component.

64-Bit ABI

Specification of a full 64-bit binary standard.

UMA

Support for the Universal Measurement Architecture.

Communications APIs

A number of requests are being evaluated.

IN Index

#ident "@(#)doctools:index/index.head 1.1"

Index

H

I

J

K

L

P

R

S

X

Y

Z