AT&T

UNIX® SYSTEM V/386 RELEASE 4

MULTIBUS® Reference Manual

UNIX Software Operation

AT&T

UNIX® SYSTEM V/386 RELEASE 4

MULTIBUS® Reference Manual

UNIX Software Operation

Published by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

ACKNOWLEDGEMENT

Portions of this book have been provided by Intel Corporation.

IMPORTANT NOTE TO USERS

While every effort has been made to ensure the accuracy of all information in this document, AT&T assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. AT&T further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. AT&T disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, *including implied warranties of merchantability or fitness for a particular purpose.* AT&T makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting or license to make, use or sell equipment constructed in accordance with this description.

AT&T reserves the right to make changes without further notice to any products herein to improve reliability, function, or design.

TRADEMARKS

UNIX is a registered trademark of AT&T.
intel, iRMK, iRMX, iSBC, iSBX, MULTIBUS, and OpenNET are registered trademarks of Intel Corporation.

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-957481-6

UNIX
PRESS
A Prentice Hall Title

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Contents

Figures and Tables

PERMUTED INDEX

Permuted Index

permissions on files	ckperms set and/or verify	ckperms(1M)
rtc MULTIBUS	clock driver	rtc(1M)
a line discipline to a line on the	communication controller /binds	ccibind(1M)
a connection to a subchannel on the	communications controller /Creates	cciattach(1M)
with a subchannel on a line on the	communications controller /host	ccidetach(1M)
a line discipline program on the	communications controller /frees	ccifree(1M)
/line discipline information on the	communications controller	ccildinfo(1M)
/line discipline list on the	communications controller	ccildlist(1M)
ccilinfo line information on the	communications controller	ccilinfo(1M)
a line discipline program on to the	communications controller /load	cciload(1M)
/CCI server information on the	communications controller	ccisrvinfo(1M)
/subchannel information on the	communications controller	ccisubinfo(1M)
/on a line to a new host on the	communications controller	cciswitch(1M)
line discipline from a line on the	communications controller /a	cciunbind(1M)
driver i410 iSBC 186/410 serial	communications controller device	i4(1M)
device driver cci	communications controller interface	cci(1M)
a line to a new host on the	communications controller /switches	ttyswitch(1M)
file format	config MULTIBUS II configuration	config(1M)
config MULTIBUS II	configuration file format	config(1M)
communications/ cciattach Creates a	connection to a subchannel on the	cciattach(1M)
	console console port device driver	console(1M)
console	console port device driver	console(1M)
i8251	console terminal/iSBXT 351 driver	i825(1M)
rci debug	console/rci protocol driver	rci(1M)
a subchannel on the communications	controller /Creates a connection to	cciattach(1M)
to a line on the communication	controller /binds a line discipline	ccibind(1M)
on a line on the communications	controller /host with a subchannel	ccidetach(1M)
program on the communications	controller /frees a line discipline	ccifree(1M)
information on the communications	controller /line discipline	ccildinfo(1M)
list on the communications	controller /line discipline	ccildlist(1M)
information on the communications	controller ccilinfo line	ccilinfo(1M)
program on to the communications	controller /load a line discipline	cciload(1M)
information on the communications	controller ccisrvinfo CCI server	ccisrvinfo(1M)
information on the communications	controller ccisubinfo subchannel	ccisubinfo(1M)
to a new host on the communications	controller /a subchannel on a line	cciswitch(1M)
from a line on the communications	controller /a line discipline	cciunbind(1M)
i258 iSBC 386/258 peripheral	controller device driver	i258(1M)
iSBC 186/410 serial communications	controller device driver i410	i4(1M)
i530 iSBC 186/530 Ethernet	controller device driver	i530(1M)
i214 iSBCR 214 peripheral	controller disk device driver	i2(1M)
i224a iSBC 186/224A peripheral	controller disk device driver	i224a(1M)
on an ethernet node9s ethernet	controller /provides information	enetinfo(1M)
resets and reloads the ethernet	controller enetload	enetload(1M)
i546 iSBC 546 multi-port serial	controller	i546(1M)
cci communications	controller interface device driver	cci(1M)
atcs asynchronous terminal	controller script device driver	atcs(1M)
i214tp iSBC 214 peripheral	controller tape device driver	i2(1M)

ramd MULTIBUS RAM	disk driver	ramd(1M)
hdformat low level hard	disk formatter	hdformat(1M)
dma	DMA device driver	dma(1M)
	dma DMA device driver	dma(1M)
library and object module formats	download loads 8086 absolute	download(1M)
terminal controller script device	driver atcs asynchronous	atcs(1M)
bps Bootstrap Parameter String	driver	bps(1M)
controller interface device	driver cci communications	cci(1M)
console console port device	driver	console(1M)
d258 i82258 ADMA device	driver	d258(1M)
dma DMA device	driver	dma(1M)
edlina External Data Link	driver for iNA961 Release 3.0	edlina(1M)
peripheral controller disk device	driver i214 iSBCR 214	i2(1M)
peripheral controller tape device	driver i214tp iSBC 214	i2(1M)
peripheral controller disk device	driver i224a iSBC 186/224A	i224a(1M)
peripheral controller tape device	driver i224atp iSBC 186/224A	i224atp(1M)
peripheral controller device	driver i258 iSBC 386/258	i258(1M)
peripheral controller tape device	driver i258tp iSBC 386/258	i258tp(1M)
354 dual channel serial-port device	driver i354 iSBX	i354(1M)
communications controller device	driver i410 iSBC 186/410 serial	i4(1M)
186/530 Ethernet controller device	driver i530 iSBC	i530(1M)
i8251 console terminal/iSBXT 351	driver	i825(1M)
iasy asynchronous (terminal) device	driver	iasy(1M)
ics Interconnect Space device	driver	ics(1M)
mpc MPC device	driver	mpc(1M)
mps Message Passing Space device	driver	mps(1M)
OSI Transport Service (ots) device	driver ots System V/386	ots(1M)
ramd MULTIBUS RAM disk	driver	ramd(1M)
rci debug console/rci protocol	driver	rci(1M)
rtc MULTIBUS clock	driver	rtc(1M)
fddd MULTIBUS flexible disk device	drivers	fddd(1M)
hddd MULTIBUS hard disk device	drivers	hddd(1M)
tdd MULTIBUS tape device	drivers	tdd(1M)
driver i354 iSBX 354	dual channel serial-port device	i354(1M)
for iNA961 Release 3.0	edlina External Data Link driver	edlina(1M)
ethernet node9s ethernet/	enetinfo provides information on an	enetinfo(1M)
ethernet controller	enetload resets and reloads the	enetload(1M)
i530 iSBC 186/530	Ethernet controller device driver	i530(1M)
information on an ethernet node9s	ethernet controller /provides	enetinfo(1M)
enetload resets and reloads the	ethernet controller	enetload(1M)
enetinfo provides information on an	ethernet node9s ethernet controller	enetinfo(1M)
iNA961 Release 3.0 edlina	External Data Link driver for	edlina(1M)
drivers	fddd MULTIBUS flexible disk device	fddd(1M)
cpout copy out	file archives to multiple volumes	cpout(1M)
config MULTIBUS II configuration	file format	config(1M)
set and/or verify permissions on	files ckperms	ckperms(1M)
interconnect space of/ ics_find_rec	finds a specific record in the	ics_find_rec(1M)

fddd MULTIBUS	flexible disk device drivers	fddd(1M)
MULTIBUS II configuration file	format config	config(1M)
absolute library and object module	formats download loads 8086	download(1M)
hdformat low level hard disk	formatter	hdformat(1M)
the communications/ ccifree	frees a line discipline program on	ccifree(1M)
dbon sets target for	front panel message delivery	dbon(1M)
string on the processor in the	given slot /the bootstrap parameter	initbp(1M)
reset resets the processor in a	given slot	reset(1M)
hddd MULTIBUS	hard disk device drivers	hddd(1M)
hdformat low level	hard disk formatter	hdformat(1M)
drivers	hddd MULTIBUS hard disk device	hddd(1M)
formatter	hdformat low level hard disk	hdformat(1M)
a subchannel on a line to a new	host on the communications/ /switch	cciswitch(1M)
ttyswitch switches a line to a new	host on the communications/	ttyswitch(1M)
the/ ccidetach disassociates this	host with a subchannel on a line on	ccidetach(1M)
controller disk device driver	i214 iSBCR 214 peripheral	i2(1M)
controller tape device driver	i214tp iSBC 214 peripheral	i2(1M)
controller disk device driver	i224a iSBC 186/224A peripheral	i224a(1M)
controller tape device driver	i224atp iSBC 186/224A peripheral	i224atp(1M)
controller device driver	i258 iSBC 386/258 peripheral	i258(1M)
controller tape device driver	i258tp iSBC 386/258 peripheral	i258tp(1M)
serial-port device driver	i354 iSBX 354 dual channel	i354(1M)
communications controller device/	i410 iSBC 186/410 serial	i4(1M)
controller device driver	i530 iSBC 186/530 Ethernet	i530(1M)
controller	i546 iSBC 546 multi-port serial	i546(1M)
d258	i82258 ADMA device driver	d258(1M)
driver	i8251 console terminal/iSBXT 351	i825(1M)
driver	iasy asynchronous (terminal) device	iasy(1M)
driver	ics Interconnect Space device	ics(1M)
record in the interconnect space/	ics_find_rec finds a specific	ics_find_rec(1M)
MULTIBUS II/ icsrd, icswr, icsslot,	icsgetrec utilities to access	icsrd(1M)
utilities to access MULTIBUS II/	icsrd, icswr, icsslot, icsgetrec	icsrd(1M)
registers of the board and returns/	ics_read reads interconnect	ics_read(1M)
access MULTIBUS II/ icsrd, icswr,	icsslot, icsgetrec utilities to	icsrd(1M)
to access MULTIBUS II/ icsrd,	icswr, icsslot, icsgetrec utilities	icsrd(1M)
registers of the board	ics_write writes into interconnect	ics_write(1M)
bootserver starts a MULTIBUS	II boot service daemon	bootserver(1M)
config MULTIBUS	II configuration file format	config(1M)
utilities to access MULTIBUS	II interconnect space /icsgetrec	icsrd(1M)
External Data Link driver for	iNA961 Release 3.0 edlina	edlina(1M)
ethernet/ enetinfo provides	information on an ethernet node9s	enetinfo(1M)
ccildinfo line discipline	information on the communications/	ccildinfo(1M)
controller ccilinfo line	information on the communications	ccilinfo(1M)
controller ccisrvinfo CCI server	information on the communications	ccisrvinfo(1M)
controller ccisubinfo subchannel	information on the communications	ccisubinfo(1M)
parameter string on the processor/	initbp initializes the bootstrap	initbp(1M)
string on the processor in/ initbp	initializes the bootstrap parameter	initbp(1M)

sgib install	MULTIBUS bootstrap loaders	sgib(1M)
rtc	MULTIBUS clock driver	rtc(1M)
drivers fddd	MULTIBUS flexible disk device	fddd(1M)
hddd	MULTIBUS hard disk device drivers	hddd(1M)
bootserver starts a	MULTIBUS II boot service daemon	bootserver(1M)
format config	MULTIBUS II configuration file	config(1M)
/icsgetrec utilities to access	MULTIBUS II interconnect space	icsrd(1M)
ramd	MULTIBUS RAM disk driver	ramd(1M)
tdd	MULTIBUS tape device drivers	tdd(1M)
cpout copy out file archives to	multiple volumes	cpout(1M)
i546 iSBC 546	multi-port serial controller	i546(1M)
/provides information on an ethernet	node9s ethernet controller	enetinfo(1M)
loads 8086 absolute library and	object module formats download	download(1M)
defect list mdl read and	optionally write the manufacturer's	mdl(1M)
driver ots System V/386	OSI Transport Service (ots) device	ots(1M)
System V/386 OSI Transport Service	(ots) device driver ots	ots(1M)
Service (ots) device driver	ots System V/386 OSI Transport	ots(1M)
dbon sets target for front	panel message delivery	dbon(1M)
bps Bootstrap	Parameter String driver	bps(1M)
initbp initializes the bootstrap	parameter string on the processor/	initbp(1M)
mps Message	Passing Space device driver	mps(1M)
i258 iSBC 386/258	peripheral controller device driver	i258(1M)
driver i214 iSBCR 214	peripheral controller disk device	i2(1M)
driver i224a iSBC 186/224A	peripheral controller disk device	i224a(1M)
driver i214tp iSBC 214	peripheral controller tape device	i2(1M)
driver i224atp iSBC 186/224A	peripheral controller tape device	i224atp(1M)
driver i258tp iSBC 386/258	peripheral controller tape device	i258tp(1M)
ckperms set and/or verify	permissions on files	ckperms(1M)
console console	port device driver	console(1M)
reset resets the	processor in a given slot	reset(1M)
bootstrap parameter string on the	processor in the given slot /the	initbp(1M)
ccifree frees a line discipline	program on the communications/	ccifree(1M)
cciload load a line discipline	program on to the communications/	cciload(1M)
rci debug console/rci	protocol driver	rci(1M)
node9s ethernet/ enetinfo	provides information on an ethernet	enetinfo(1M)
ramd MULTIBUS	RAM disk driver	ramd(1M)
	ramd MULTIBUS RAM disk driver	ramd(1M)
driver	rci debug console/rci protocol	rci(1M)
manufacturer's defect list mdl	read and optionally write the	mdl(1M)
board and returns the/ ics_read	reads interconnect registers of the	ics_read(1M)
a/ ics_find_rec finds a specific	record in the interconnect space of	ics_find_rec(1M)
the/ ics_read reads interconnect	registers of the board and returns	ics_read(1M)
ics_write writes into interconnect	registers of the board	ics_write(1M)
Data Link driver for iNA961	Release 3.0 edlina External	edlina(1M)
enetload resets and	reloads the ethernet controller	enetload(1M)
given slot	reset resets the processor in a	reset(1M)
controller enetload	resets and reloads the ethernet	enetload(1M)

slot reset	resets the processor in a given	reset(1M)
/registers of the board and	returns the values in the buffer	ics_read(1M)
	rtc MULTIBUS clock driver	rtc(1M)
asynchronous terminal controller	script device driver atcs	atcs(1M)
device driver i410 iSBC 186/410	serial communications controller	i4(1M)
i546 iSBC 546 multi-port	serial controller	i546(1M)
i354 iSBX 354 dual channel	serial-port device driver	i354(1M)
communications/ ccisrvinfo CCI	server information on the	ccisrvinfo(1M)
starts a MULTIBUS II boot	service daemon bootserver	bootserver(1M)
ots System V/386 OSI Transport	Service (ots) device driver	ots(1M)
files ckperms	set and/or verify permissions on	ckperms(1M)
delivery dbon	sets target for front panel message	dbon(1M)
loaders	sgib install MULTIBUS bootstrap	sgib(1M)
on the processor in the given	slot /bootstrap parameter string	initbp(1M)
resets the processor in a given	slot reset	reset(1M)
ics Interconnect	Space device driver	ics(1M)
mps Message Passing	Space device driver	mps(1M)
to access MULTIBUS II interconnect	space /icsslot, icsgetrec utilities	icsrd(1M)
specific record in the interconnect	space of a board /finds a	ics_find_rec(1M)
space of a/ ics_find_rec finds a	specific record in the interconnect	ics_find_rec(1M)
daemon bootserver	starts a MULTIBUS II boot service	bootserver(1M)
bps Bootstrap Parameter	String driver	bps(1M)
/initializes the bootstrap parameter	string on the processor in the/	initbp(1M)
communications/ ccisubinfo	subchannel information on the	ccisubinfo(1M)
/disassociates this host with a	subchannel on a line on the/	ccidetach(1M)
on the/ cciswitch switch a	subchannel on a line to a new host	cciswitch(1M)
cciattach Creates a connection to a	subchannel on the communications/	cciattach(1M)
new host on the/ cciswitch	switch a subchannel on a line to a	cciswitch(1M)
the communications/ ttyswitch	switches a line to a new host on	ttyswitch(1M)
(ots) device driver ots	System V/386 OSI Transport Service	ots(1M)
iSBC 214 peripheral controller	tape device driver i214tp	i2(1M)
iSBC 186/224A peripheral controller	tape device driver i224atp	i224atp(1M)
iSBC 386/258 peripheral controller	tape device driver i258tp	i258tp(1M)
tdd MULTIBUS	tape device drivers	tdd(1M)
delivery dbon sets	target for front panel message	dbon(1M)
	tdd MULTIBUS tape device drivers	tdd(1M)
driver atcs asynchronous	terminal controller script device	atcs(1M)
iasy asynchronous	(terminal) device driver	iasy(1M)
i8251 console	terminal/iSBXT 351 driver	i825(1M)
mpscnv Convert	transport code	mpscnv(1M)
driver ots System V/386 OSI	Transport Service (ots) device	ots(1M)
host on the communications/	ttyswitch switches a line to a new	ttyswitch(1M)
line on the/ cciunbind	unbinds a line discipline from a	cciunbind(1M)
icsrd, icswr, icsslot, icsgetrec	utilities to access MULTIBUS II/	icsrd(1M)
device driver ots System	V/386 OSI Transport Service (ots)	ots(1M)
of the board and returns the	values in the buffer /registers	ics_read(1M)
ckperms set and/or	verify permissions on files	ckperms(1M)

copy out file archives to multiple	volumes cpout	cpout(1M)
list mdl read and optionally	write the manufacturer's defect	mdl(1M)
of the board ics_write	writes into interconnect registers	ics_write(1M)

1. OVERVIEW

1 Manual Overview

Manual Overview

This manual lists and describes commands and device drivers Intel® Corporation has added to the System V/386 operating system for MULTIBUS support. These commands and drivers have been added to support the Intel installation procedures for over-install and add-on products, to make system administration simpler, and to interface with Intel devices.

This manual is designed to supplement the UNIX® System V/386 Release 4.0 Programmer's Reference Manual, and the UNIX System V/386 Release 4.0 User's/System Administrators Reference Manual.

This manual is divided into eight chapters and three appendices:

Chapter 1. Introduction
Provides a manual overview, explains the organization, and describes notational conventions.

Chapter 2. Commands
Alphabetically lists System V/386 MULTIBUS commands.

Chapter 3. Functions and Libraries
Lists System V/386 MULTIBUS functions and libraries.

Chapter 4. File Formats
Lists System V/386 MULTIBUS file formats.

Chapter 5. Device Information
Provides information for System V/386 MULTIBUS devices.

Chapter 6. Device Drivers
Alphabetically lists System V/386 MULTIBUS device drivers.

Chapter 7. System Maintenance Commands
Alphabetically lists Intel's System V/386 system maintenance commands.

Chapter 8. Static Bad Block Handling
Describes those routines that compensate for the unusable blocks of a hard disk drive.

Appendix A.
iSXMT 279 Window Interface

Organization

All entries are based on a format common to other UNIX reference manuals. Not every section appears for every command.

- The NAME section gives the name(s) of the entry and briefly states its purpose.
- The SYNOPSIS section summarizes the use of the command.
- The DESCRIPTION section describes the way the program works.
- The EXAMPLES section gives examples of usage, where appropriate.
- The FILES section lists files that are built into or affected by the command.
- The NOTES section provides additional information regarding the use of the command.
- The SEE ALSO section provides a list of other relevant commands.

Notational Conventions

The following notational conventions are used throughout this manual:

input	User input, such as commands, options, and arguments to commands, variables, and names of directories and files, appear in bold. In user-entry instructions, `Enter wd:` means type the characters wd, then press <RETURN>.
output	System output, such as prompt signs and responses to commands, appear in constant width type.

variable	Names of variables to which values must be assigned (such as filename) appear in italic.
command (#) or (# iref)	Refers to a command, library call, or system call. The numbers follow the convention of the AT&T manuals. A command name followed by (#) refers to a command in the AT&T manual set , or (# iref) refers to other references that are in this manual.

2. COMMANDS

2 MULTIBUS Commands

MULTIBUS I and MULTIBUS II Commands

This chapter lists Intel's System V/386 commands for MULTIBUS I and MULTIBUS II. They are:

- ckperms
- cpout
- hdformat
- mdl
- sgib

Commands that apply only to MULTIBUS II are:

- bootserver
- cciattach
- ccibind
- ccidetach
- ccifree
- ccildinfo
- ccildlist
- ccilnfo
- cciload
- ccisrvinfo
- ccisubinfo
- cciswitch
- cciunbind
- download
- enetinfo
- enetload
- mpscnv
- ttyswitch

NAME

`ckperms` – set and/or verify permissions on files

SYNOPSIS

`ckperms [-i` *idlist*`] [-c] [-g] [-v ] [-C]`
`[-S] [-t` *flist*`] [-n` *pathname* `[-l` *logfile*`]`
perms.filename

DESCRIPTION

The `ckperms` utility reads system file information from a product definition file, also called a perms file. The capabilities of `ckperms` vary depending on which options are specified.

`-i`	Processes only those files whose package identifier matches the list of idnames specified in *idlist*. The *idlist* is a list of idnames separated by commas with no spaces.
`-c`	Specifies that the checksums are to be verified. The checksum field of a block special or character special file represents the major and minor numbers, and is verified even when the `-c` option is not specified.
`-g`	Prints to standard output a list of pathnames being processed. Files not included in this list of pathnames are: block special files, character special files, directory files, empty files, and pipe files.
`-v`	Prints to standard output a list of all pathnames being processed.
`-C`	Changes the characteristics of files in the system to match the specifications in the database, access and ownership permissions are set, and type)e9 files, directories, special files, and links are created.
`-S`	Strips specified files.
`-t`	Processes files whose file type matches those specified in *flist*. The entries in *flist* consist of file types separated by commas with no spaces.
-n	Provides a pathname to be added at the beginning of all relative pathnames in the database. If this option is not specified, all relative pathnames start at the current working directory.
`-l`	Logs the characteristics of the files after the files have been processed.

FILES

`/sbin/ckperms`

NOTES

Exit Codes

0	Returned on successful completion
1	Returned if there are errors in the database
2	Returned in case of errors in command line syntax
3	Returned in case of mismatches between the database and the file processed

NAME

cpout – copy out file archives to multiple volumes

SYNOPSIS

`cpout [-a] [-c] [-B] [-v] [-V] [-k]`
`[-C` *bufsize*`] [-M` *message*`]` *nblks specfile*

DESCRIPTION

The cpout program reads the standard input to obtain a list of pathnames and then copies those files into the file specified in *specfile*. The output is written in cpio format. The output medium is assumed to be nblks blocks in size, where a block is 512 bytes if the -k option is not specified, and 1024 bytes if the -k option is specified. When cpout finishes writing the number of blocks specified in nblks, it prints the message:

```
Insert Volume #: <RETURN> to continue, s <RETURN> to skip
```

To continue, replace the medium and press <RETURN>. Typing
s <RETURN> causes cpout to skip to the next volume of the
output medium. Each volume becomes an independent cpio archive.
The above message also appears as a prompt before beginning to write the
first volume of the output medium.

The options are:

-a	Resets access time of input files after they have been copied.
-c	Writes header information in ASCII format for portability. (Always use this option when the origin and destination machines are different types.)
-B	Blocks output 5,120 bytes per record (meaningful only with data directed to a character special device.)
-v	Prints a list of file names as they are processed.
-V	Special verbose mode. This option causes a dot to be printed for each file seen.
-k	Sets the block size to 1024 bytes. Default is 512 bytes.
-C	Causes the input/output to be blocked bufsize bytes per record. *bufsize* must be a positive integer. The default buffer size is 512 bytes when the -C and -B options are not specified. The -C option is meaningful only with data directed to or from a character special file (for example, /dev/rmt/c0s0).
-M	Used to define a message to be displayed when the end of an output medium is reached. Data inconsistencies are introduced when the end of the physical medium is reached. Thus, if this message appears, it indicates that data corruption may have occurred.

FILES

/sbin/cpout

NOTES

Upon completion, the cpout command writes the number of output volumes processed to stderr.

The *specfile* argument cannot be an ordinary file. Normally, *specfile* will be the pathname of a special device file such as a tape drive or a floppy disk drive. This must be specified as the raw device.

Reaching the end of the physical medium before nblks blocks are written is not allowed and results in inconsistent data written to the output medium. Therefore, the value of nblks must always be less than the actual number of blocks that the output medium can contain.

NAME

`hdformat` – low level hard disk formatter.

SYNOPSIS

`hdformat` [`-c` *cylinders*] [`-d` *bytes/sector*]
[`-f` *heads*] [`-s sectors/track`] [`-i` *interleave*] *character-device*

DESCRIPTION

The `hdformat` command is a low level disk format program. It formats the drive specified by *character-device* with instructions to the disk driver to map the manufacturer's defect list. **`hdformat`** formats slice zero of the drive and should be used after a valid manufacturer's defect list has been installed on the drive (see `mdl`(1iref)).

This operation destroys the contents of the specified disk.

Specify character-device as a device file from the `/dev` directory.

Command line options are:

`-c`	The number of cylinders the drive contains
`-d`	The bytes per sector or density of the drive
`-f`	The number of fixed heads the drive contains
`-s`	Sectors per track of the drive
`-i`	The interleave of the drive

FILES

`/sbin/hdformat`

SEE ALSO

`mdl(1iref)`, `mdl(1iref)`

NAME

mdl – read and optionally write the manufacturer's defect list.

SYNOPSIS

`mdl` [`-N` `-A`] `-c` *cylinders* `-d` *bytes/sector*
`-f` *heads* `-s` *sectors/track* [`-b`] *character-device*

DESCRIPTION

The `mdl` utility, without a `-A` or `-N` flag, attempts to read and display the manufacturer's defect list from the disk specified by *character-device*. If the defect list is empty, `mdl` asks the user to add any bad tracks from the printed copy of the list obtained with the drive, or reported by the system. `mdl` writes this information to the defect list.

This utility may be invoked at any time to view or optionally update the defect list. However, the new or updated defect list becomes effective only after `hdformat` is executed.

Specify *character-device* as a device file from the /dev directory. Return values are 0 for successful completion and non-zero for failure. Command line options are as follows:

`-N`	Write a new defect list over one that has become corrupted.
`-A`	Append new information to the defect list.
`-c`	The number of cylinders the drive contains.
`-d`	The bytes per sector or density of the drive.
`-f`	The ammount of fixed heads the drive contains.
`-s`	Sectors per track of the drive.
`-b`	Do the operation in batch mode. The format of the output to stdout is the format of the input to stdin.

NOTES

The user can add to the bad track list. Then, at the next `hdformat`, these additional bad tracks will be hardware mapped to alternates rather than software mapped, thus reducing system overhead.

FILES

`/sbin/mdl`
`/sbin/hdformat`

NAME

`sgib` – install MULTIBUS bootstrap loaders

SYNOPSIS

`sgib` [`-B` ? `-D` ?`-F` ? `-T` ?`-R`]
[`-d` *granularity*] [`-i` *interleave*]
[`-f` *fheads*] [`-o` *offset*] [`-r` *rheads*]
[`-c` *cyl*] [`-s` *sec*] [`-M` *filename*]
[`-N` **filename**] *device_name*

DESCRIPTION

`sgib` installs the system boot block and boot strap images on either hard disk or tape devices, as specified by the device_name parameter, for MULTIBUS I and MULTIBUS II systems. Bootloaders for both MSA (MULTIBUS II System Architecture) and non-MSA are installed if *device_name* specifies a hard disk. The defaults are for a 140 megabyte Maxtor hard disk drive. The default bootloader image for the non-MSA portion is `/etc/boot`, and the default image for the MSA bootloader is `/etc/dsboot`.

The options are:

`-B`	Produce a file suitable for use by the MULTIBUS II bootserver.
`-D`	Produce a file suitable for use on a hard disk. This is the default.
`-F`	Produce a file suitable for use on a floppy disk.
`-T`	Produce a file suitable for use on a boot tape.
`-R`	Real mode boot flag.
`-d` *granularity*	Specify the number of bytes per sector.
`-i` *interleave*	Specify the format interleave factor.
`-f` *fheads*	Specify the number of fixed heads.
`-o` *offset*	Specify the offset in sectors to the start of the root file system.
`-r` *rheads*	Specify the number of removable heads on the drive. This is normally 0 for hard disks.
`-c` *cyl*	Specify the number of cylinders. This should include cylinders which are reserved for manufacture's defect lists, and system diagnostics.
`-s` *sec*	Specify the number of sectors per track.
`-M` *filename*	Specify the name of the MSA boot file.
`-N` *filename*	Specify the name of the non-MSA boot file.

FILES

```
/sbin/sgib
/etc/boot
/etc/dsboot
/etc/tsboot
/etc/bsboot
```

NOTES

`sgib` should be used on the raw device.

DIAGNOSTICS

0 is returned upon successful completion. Exit status of 1 is returned in the event of errors.

SEE ALSO

`ivlab`(4)

NAME

`bootserver` – starts a MULTIBUS II boot service daemon

SYNOPSIS

`bootserver` [`-c` *configfile*] [`-l` *logfile*] [`-x` *loglevel*]

DESCRIPTION

`bootserver` starts a booting service for MULTIBUS II. Once invoked, `bootserver` disconnects itself from the invoking terminal and process, opens its MULTIBUS II Transport listening port (0x500), and listens for booting and configuration requests from other processor hosts on MULTIBUS II. The information given for the requests is specified in the configuration file. Refer to the MULTIBUSR II System Architecture Bootstrap Specification for exact details. Command line options are:

`-c` *configfile*
: The specified file is used as the configuration file. If no configuration file is specified, the bootserver asks the **bps**(7iref) driver for the name of the configuration file specified by the parameter `BL_config_file`. If that fails, the `/etc/default/bootserver/config` file is used. Refer to `config`(4iref) for the format of the configuration file. The configuration file is reread if it is modified after the bootserver is started.

`-l` *logfile*
: Activity information is written to the specified *logfile*. Logging information is written as text lines that begin with a date/time stamp followed by the message. No effort is made to restrict the size of the *logfile*. If no *logfile* parameter is given, the activity information will be written to standard output.

`-x` *loglevel*
: Controls the amount of information written to the *logfile*. The *loglevel* can be one of the following decimal values:
0 = error logging only
1 = error and connect information (default)
2 = level 1 plus very detailed communication information

If loglevel 1 is specified, information on what is being requested from the boot server and who is requesting it is reported. If no `-x` parameter is specified, error and connection information is written to the *logfile* (as if loglevel 1 had been specified).

FILES

`/etc/default/bootserver/config`

SEE ALSO

`config`(4iref)

NOTES

Parameters that are duplicated in a host configuration line and in the global configuration line are duplicated in the boot parameter string that is returned to the host.

NAME

`cciattach` – Creates a connection to a subchannel on the communications controller.

SYNOPSIS

`/usr/lib/cci/cciattach` [–p *portid*] *slot-ID line-number sub-channel*

DESCRIPTION

`cciattach` associates a subchannel on a previously bound line with this host on the communications controller. This program sends a CCI_Attach message to the CCI server. If at the time the CCI_Attach message is received by the CCI server and the subchannel is already attached by another host, the response will be delayed until the other host relinquishes control of the subchannel and control eventually comes back to this host. When the response arrives, a MULTIBUS II Transport portID will be returned for the subchannel. The issuing host may now send line discipline specific messages to the line discipline program directed at the subchannel using this port ID. This host remains the active host on the subchannel until it gives up control either by using the `ccidetach`(1iref) command or the `cciswitch`(1iref) command.

Command line options are:

–p *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff. This value is also used in the default_portid field of the CCI_Attach message.

The host can receive a response to the CCI_Attach message as a result of another host sending a `cciswitch`(1iref) or a `ccidetach`(1iref) command to the CCI server. In this case, the CCI_Attach response will indicate this fact and will contain the host ID of the previous host. The response also contains the line discipline specific information returned from the line discipline program and the status of the session on the subchannel from the previous host.

The host issuing this command must have previously issued CCI_Bind message using the `ccibind`(1iref) command to register itself as a potential user of the line. Otherwise an error will be returned from the CCI server.

The *line-number* can range from 0 to (MAX_LINES ? 1), where MAX_LINES is returned in the `ccisrvinfo`(1iref) command. The subchannel can range from 0 to (MAX_SUBCHANNELS ? 1), where MAX_SUBCHANNELS is returned in the `ccibind`(1iref) command. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

`/usr/lib/cci/cciattach`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccifree`(1iref), `cciunbind`(1iref), `cciload`(1iref), `ccidetach`(1iref), `cciswitch`(1iref), `cci`(7iref)

NAME

`ccibind` – binds a line discipline to a line on the communication controller.

SYNOPSIS

`/usr/lib/cci/ccibind` [–p *portid*] *slot-ID line-number line-discipline-ID*

DESCRIPTION

`ccibind` associates a line with a previously loaded line discipline program on the communications controller. The line could be either a physical serial communications line on the controller or it could be used to associate with a job on the controller. The interpretation of the line is left to the line discipline program on the controller. In addition, this host is registered as a potential user of the line. This program sends a CCI_Bind message to the CCI server. The response to this command will contain the maximum subchannels supported on this line by the line discipline and this will be displayed by the command on the issuing host. The host issuing this command must have previously issued a CCI_Create message using the `cciload`(1iref) command to set up the line discipline on the controller. Otherwise an error will be returned from the CCI server.

Command line options are:

–p *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

A host can bind a line to a *line-discipline-ID* only once. The first CCI_Bind message issued for the line causes a job to be created for the line. Subsequent CCI_Bind messages from other hosts for the same line and *line-discipline-ID* will cause these hosts to be added to the set of potential users of the line. If one of these messages contains a *line-discipline-ID* that is different from the *line-discipline-ID* bound to the line, an error response will be returned by the CCI server.

The *line-discipline-ID* specified is associated with a line discipline image on the controller. The *line-discipline-ID* can range from 0 to 255. The *line-number* can range from 0 to MAX_LINES - 1, where MAX_LINES is returned in the `ccisrvinfo`(1iref) command. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

`/usr/lib/cci/ccibind`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccifree`(1iref), `cciunbind`(1iref), `cciload`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref), `cci`(7iref)

NAME

`ccidetach` – disassociates this host with a subchannel on a line on the communications controller.

SYNOPSIS

`/usr/lib/cci/ccidetach` [`-p` *portid*] *slot-ID line-number subchannel*

DESCRIPTION

`ccidetach` cancels the association set up between a *subchannel* and a host on the communications controller. The host must have sent a CCI_Attach message for the *subchannel* using the `cciattach`(1iref) command. This program sends a CCI_Detach message to the CCI server.

Command line options are:

`-p` *n*	Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

If there are other hosts queued at the *subchannel*, the first host on the queue is made active on the `subchannel`.
The *line-number* can range from 0 to MAX_LINES - 1, where MAX_LINES is returned in the `ccisrvinfo`(1iref) command. The `subchannel` can range from 0 to MAX_SUBCHANNELS - 1, where MAX_SUBCHANNELS is returned in the `ccibind`(1iref) command. The communications controller is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

`/usr/lib/cci/ccidetach`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccifree`(1iref), `cciunbind`(1iref), `cciload`(1iref), `cciattach`(1iref), `cciswitch`(1iref), `cci`(7iref)

NAME

`ccifree` – frees a line discipline program on the communications controller.

SYNOPSIS

`/usr/lib/cci/ccifree` [-p *portid*] *slot-ID line-discipline-ID*

DESCRIPTION

`ccifree` frees the memory used by a line discipline program which was previously downloaded to the communications controller using the `cciload`(1iref) command. This command must be issued only after the host has unbound all the lines bound to the line discipline on the controller using the `cciunbind`(1iref) commands. This program sends a CCI_Free message to the CCI server on the controller.

Command line options are:

-p *n* Set the MULTIBUS II Transport source Portid to n instead of the default value 0x1ff.

The CCI server will only release the memory used for the line discipline program when all hosts that have issued CCI_Create messages for the script have issued corresponding CCI_Free messages.

The host can issue a CCI_Free message any time after it has issued a CCI_Create message. If it is used before the host receives a response to the CCI_Create message, it will be used to cancel the previously issued CCI_Create message. In this case, there will be an error response to the CCI_Create message followed by the response to the CCI_Free message.

The *line-discipline-ID* specified is associated with a line discipline on the CCI server. The *line-discipline-ID* can range from 0 to 255. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/ccifree`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `cciload`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref), `cci`(7iref)

NAME

`ccildinfo` – line discipline information on the communications controller.

SYNOPSIS

`/usr/lib/ccildinfo` [`-p` *portid*] *slot-ID line-discipline-ID*

DESCRIPTION

`ccildinfo` displays information about a line discipline on the communications controller. The program displays the state of the line discipline on the controller. The state can be either not present, being downloaded or present. The number of hosts that have issued CCI_Create messages is displayed along with their host IDs. The program also displays the number of lines bound to this line discipline along with the line numbers that are bound to this line discipline.

Command line options are:

-p *n* Set the MULTIBUS II Transport source Portid to n instead of the default value 0x1ff.

The program sends a CCI_Get_Line_Discipline_Info message to the CCI server and prints the information obtained in the reply from the server. The line discipline is specified using its *line-discipline-ID* on the CCI server. This value can range from 0 to 255. The communications controller hosting the CCI server is selected by specifying the MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/ccildinfo`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `cciload`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `ccifree`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

`ccildlist` – line discipline list on the communications controller.

SYNOPSIS

`/usr/lib/cci/ccildlist` [-p *portid*] *slot-ID*

DESCRIPTION

`ccildlist` displays the line discipline IDs of the line disciplines resident on the communications controller.

Command line options are:

-p *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

The program sends a CCI_Get_Line_Discipline_List message to the CCI server and prints the information obtained in the reply from the server. The program displays the number of line disciplines resident and their line Discipline IDs. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/ccildlist`

SEE ALSO

`ccisrvinfo`(1iref), `ccildinfo`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `cciload`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `ccifree`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

`ccilinfo` – line information on the communications controller

SYNOPSIS

`/usr/lib/cci/ccilinfo` [`-p` *portid*] *slot-ID line-number*

DESCRIPTION

`ccilinfo` displays information about a line on the communications controller. The program displays the state of the line on the controller. The state can be either bound or not bound to a line discipline. If the line is bound to a line discipline, the line discipline ID of the line discipline is displayed. The number of hosts that have issued CCI_Bind messages is displayed along with their host IDs. The program also displays the maximum number of subchannels supported on this line by the line discipline.

Command line options are:

`-p` *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

The program sends a CCI_Get_Line_Info message to the CCI server and prints the information obtained in the reply from the server. The line is specified using its line-number on the CCI server. The lines are numbered starting from 0. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

`/usr/lib/cci/ccilinfo`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccildinfo`(1iref), `ccisubinfo`(1iref), `cciload`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `ccifree`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

`cciload` – load a line discipline program on to the communications controller.

SYNOPSIS

`/usr/lib/cci/cciload` [-p *portid*] [-v] *slot-ID* *line-discipline-ID line-discipline-filename*

DESCRIPTION

`cciload` reads a line discipline program file and sends it to the CCI server on the communications controller. The line discipline program is sent to the CCI server only if the line discipline is not present on the controller. The program obtains this information in the reply to the CCI_Create message sent to the CCI server. The state of the line discipline on the controller can be obtained using the `ccildlist`(1iref) or the `ccildinfo`(1iref) commands.

Command line options are:

-p *n* Set the MULTIBUS II transport source port-id to *n* instead of the default value 0x1ff.

-v Verbose. Normally, the line discipline is loaded silently. This option will print trace information while the line discipline is being loaded.

The program first issues a CCI_Create message and checks the status of the reply from the CCI Server. If the status indicates that the line discipline already exists on the communications controller, a message is printed to that effect and the program exits. If the status indicates that the line discipline does not exist, the program will read the specified line discipline file and send it to the controller by issuing CCI_Download messages. If an error occurs while downloading the line discipline, the program will issue a CCI_Free message to the CCI server and exit. The *line-discipline-ID* specified is associated with line discipline image on the CCI server. The *line-discipline-ID* can range from 0 to 255. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID. The line discipline file must be in OMF86 format. Only Load-Time-Locatable (LTL) or Position-Independent-Code (PIC) OMF86 formats are supported. Overlays are not supported.

FILES

`/usr/lib/cci/cciload`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `ccifree`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

ccisrvinfo – CCI server information on the communications controller.

SYNOPSIS

/usr/lib/cci/ccisrvinfo [-p *portid*] *slot-ID*

DESCRIPTION

ccisrvinfo displays the maximum number of physical serial lines available on the communications controller and the version number of the CCI server running on the controller.

Command line options are:

-p *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

The program sends a CCI_Get_Server_Info message to the CCI server and prints the information obtained in the reply from the server. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

/usr/lib/cci/ccisrvinfo

SEE ALSO

ccildlist(1iref), ccildinfo(1iref), ccilinfo(1iref), ccisubinfo(1iref), cciload(1iref), ccibind(1iref), cciunbind(1iref), ccifree(1iref), cciattach(1iref), ccidetach(1iref), cciswitch(1iref)

NAME

`ccisubinfo` – subchannel information on the communications controller.

SYNOPSIS

`/usr/lib/cci/ccisubinfo` [–p *portid*] *slot-ID line-number sub-channel*

DESCRIPTION

`ccisubinfo` displays information about the subchannel on a line on the communications controller. The program displays the state of the subchannel on the controller. The state can be either attached, or not attached to a host. If the subchannel is attached to a host, the host ID of the active host is displayed. The number of hosts that are queued on the subchannel (i.e., the number of hosts other than the active host that have issued a CCI_Attach or a CCI_Switch message) is displayed along with their host IDs. The program also displays the MULTIBUS II Transport Port ID assigned to the subchannel, for sending line discipline specific messages, if the subchannel is attached to a host.

Command line options are:

–p *n* Set the MULTIBUS II transport source port-id to *n* instead of the default value 0x1ff.

The program sends a CCI_Get_Subchannel_Info message to the CCI server and prints the information obtained in the reply from the server. The line is specified using its line-number on the controller. The lines are numbered starting from 0. The subchannel is specified by its number. The subchannels are numbered starting from 0. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/ccisubinfo`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccildinfo`(1iref), `ccilinfo`(1iref), `cciload`(1iref), `ccibind`(1iref), `cciunbind`(1iref), `ccifree(liref)`, `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

`cciswitch` – switch a subchannel on a line to a new host on the communications controller.

SYNOPSIS

`/usr/lib/cci/cciswitch` [**-p** *portid*] *slot-ID line-number sub-channel new-host-ID*

DESCRIPTION

`cciswitch` causes the issuing host to temporarily relinquish control of a subchannel on the communications controller to another host. This program sends a CCI_Switch message to the CCI server. If at the time the CCI_Switch message is received by the server and the subchannel is already attached by the specified new host, the response for the CCI_Switch message will be delayed until the other host relinquishes control of the subchannel and control eventually comes back to this host. Otherwise an error response is returned immediately.

Command line options are:

-p *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

When this host gets control of the subchannel, the response will contain:
the host ID of the previous host
the line discipline specific information (returned from the line
discipline program running on the line)
the status of the session on the subchannel from the previous host
The host issuing this command must have previously issued a CCI_Attach message (using the `cciattach`(1iref) command) to be the active host on the subchannel. Otherwise an error will be returned from the CCI server.
The line-number can range from 0 to MAX_LINES - 1, where MAX_LINES is returned in the `ccisrvinfo`(1iref) command. The sub-channel can range from 0 to MAX_SUBCHANNELS - 1, where MAX_SUBCHANNELS is returned in the `ccibind`(1iref) command. The new-host-ID is specified using its MULTIBUS II iPSB *slot-ID*. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/cciswitch`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref),
`ccisubinfo`(1iref), `ccildinfo`(1iref), `ccifree`(1iref),
`cciunbind`(1iref), `cciload`(1iref), `ccidetach`(1iref),
`cciattach`(1iref)

NAME

cciunbind – unbinds a line discipline from a line on the communications controller.

SYNOPSIS

`/usr/lib/cci/cciunbind` [`-p` *portid*] *slot-ID line-number*

DESCRIPTION

`cciunbind` deletes the association established between a line, a line discipline and this host by an earlier `ccibind` (1iref) command. The program sends a CCI_Unbind message to the CCI server.

This command must be issued only after all the subchannels on the line attached to this host have been detached. Otherwise an error will be returned from the CCI server.

Command line options are:

`-p` *n* Set the MULTIBUS II Transport source Portid to *n* instead of the default value 0x1ff.

Only the last CCI_Unbind message issued for the line causes the job running on the line to be deleted. The other CCI_Unbind messages from other hosts for the same line will cause these hosts to be removed from the set of potential users of the line.

The *line-number* can range from 0 to MAX_LINES - 1, where MAX_LINES is returned in the `ccisrvinfo`(1iref) command. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB *slot-ID*.

FILES

`/usr/lib/cci/cciunbind`

SEE ALSO

`ccisrvinfo`(1iref), `ccildlist`(1iref), `ccilinfo`(1iref), `ccisubinfo`(1iref), `ccildinfo`(1iref), `ccifree`(1iref), `ccibind`(1iref), `cciload`(1iref), `cciattach`(1iref), `ccidetach`(1iref), `cciswitch`(1iref)

NAME

`download` – loads 8086 absolute library and object module formats

SYNOPSIS

`/usr/lib/cci/download` [`-b` *buffer size*] [`-t` *time delay*]
[`-p` *portid*] [`-lrv`] *file-name slot-ID*

DESCRIPTION

`download` loads Absolute OMF86 and LIB86 format files onto a target board having the Download Protocol Routine resident in firmware (such as the iSBCR 186/410 and the iSBC 186/530). An executable memory image of the file is constructed in the local memory of the target. After loading, control on the target is transferred to the start address specified within the file (unless the `-l` option is specified).

The program opens and reads the specified file and interprets the OMF. The image is transferred to the target using MULTIBUS II Transport.

Command line options are:

`-b`*n*	Set the temporary buffer size to *n* bytes instead of the default 32,000 bytes.
`-t`*n*	Set the number of milliseconds to wait between transport messages used to transfer the image to the target. The default is 10.
`-p`*n*	Set the MULTIBUS II Transport portID for this host to *n* instead of the default 36864.
`-l`	Load only. Loads the specified file onto the target, but does not transfer control or begin execution.
`-r`	Reset. Performs a local reset to the target board before loading begins.
`-v`	Verbose. Normally, download does its work silently. This option causes each module name to be printed as it is loaded.

The file must be in Absolute OMF86 or LIB86 format. Overlays are not supported. The target board is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES

`/usr/lib/cci/download`	
`/etc/default/download`	contains the default time delay

SEE ALSO

For a definition of the Download Protocol Routine required in firmware (to work with download), see one of the following manuals.

iSBCR 186/410 Hardware Reference Manual
iSBCR 186/530 Hardware Reference Manual

NAME

`enetinfo` – provides information on an ethernet node's ethernet controller

SYNOPSIS

`/usr/sbin/enetinfo` [-a] [-b] [-e] [-h]
[-i] [-l] [-n] [-t] [-r]

DESCRIPTION

The `enetinfo` utility displays information concerning the ethernet transport drivers installed on the local node. The utility will return an error if the driver is not installed or if the system is not booted on a networked kernel.

If no option is specified, `enetinfo` displays the 12-character Ethernet address.

Command line options are:

-a	Returns all information generated by the other command line options.
-b	Returns the number of ethernet controller boards configured in the system.
-e	Returns the 12-character Ethernet address (default display).
-h	Displays a help message.
-i	Returns the MULTIBUS II slot number used by the controller board
-l	Returns the base board address used to load iNA 961 to the board.
-n	Returns the maximum number of virtual circuits (VCs) supported.
-t	Returns the type of board providing ethernet services (for example, 186/530).
-r	Returns the iNA 961 version resident on the board.

NAME

enetload – resets and reloads the ethernet controller

SYNOPSIS

/usr/sbin/enetload [-v *version*] [-b *bus_type*]

DESCRIPTION

The enetload utility resets and reloads the Ethernet controller. If enetload is invoked without any options, the system displays a dynamic menu of available iNA 961 downloadable files (based on input records from the file /etc/ina961.data). The user then selects the iNA 961 file to be downloaded.

enetload changes the network environment to use the specified version of iNA 961. Both the current operating environment and the default environment invoked at system startup are updated.

enetload performs the following functions:

1. Terminates all existing virtual circuits (VCs).
2. Resets the Ethernet controller.
3. Downloads the requested version of iNA 961 to the Ethernet controller.

Command line options are:

–v *version* Specifies the version number of the iNA 961 file to be downloaded. This file is located in the /etc directory and must have the following format:

```
ina961.version
```

–b *bus_type*
Specifies the type of system being used. Only MULTIBUS II (iSBC 186/530) is supported, so *bus_type* must be MB2.

FILES

/etc/ina961.data

SEE ALSO

enetinfo(1iref), i530(7iref), edlina(7iref)

NAME

mpscnv – Convert transport code

DESCRIPTION

The mpscnv utility reads C source code presented on its standard input looking for identifiers associated with Multibus II Interconnect Space or Message Passing Space access. These identifiers are replaced with their equivalents, as described in the *System V/386 Device Driver Interface* specification, as the file is copied to the standard output.

This utility is intended as a migration aide for porting System V/386 Release 3.2 device drivers to Release 4.0 and will not be provided in the next release.

The following lists the old identifiers and their equivalents for System V.4.

Release 3.2 Identifier	**Release 4.0 Identifier**
AMPcancel	mps_AMPcancel
AMPreceive	mps_AMPreceive
AMPreceive_frag	mps_AMPreceive_frag
AMPsend	mps_AMPsend
AMPsend_data	mps_AMPsend_data
AMPsend_reply	mps_AMPsend_reply
AMPsend_rsvp	mps_AMPsend_rsvp
BLKPRIO	MPS_BLKPRIO
BOARD_SPECIFIC_REC	ICS_BOARD_ SPECIFIC_REC
CLKPRIO	MPS_CLKPRIO
EOT_TYPE	ICS_EOT_TYPE
FP_PORT	MPS_FP_PORT
HI	ICS_HI
HIB	MPS_HIB
HOST_ID_TYPE	ICS_HOST_ID_TYPE
ICBistDataIn	ICS_BistDataIn
ICBistDataOut	ICS_BistDataOut
ICBistMasterStatus	ICS_BistMasterStatus
ICBistSlaveStatus	ICS_BistSlaveStatus
ICBistSupportLevel	ICS_BistSupportLevel
ICBistTestID	ICS_BistTestID
ICClassID	ICS_ClassID
ICGeneralControl	ICS_GeneralControl
ICGeneralStatus	ICS_GeneralStatus
ICHardwareTestRev	ICS_HardwareTestRev
ICNMIEnable	ICS_NMIEnable
ICProductCode	ICS_ProductCode
ICProgramTableIndex	ICS_ProgramTableIndex
ICVendorIDH	ICS_VendorIDH
ICVendorIDL	ICS_VendorIDL
IC_DEBUG_MASK	ICS_DEBUG_MASK
IC_DEBUG_NMI	ICS_DEBUG_NMI
IC_ENABLE_NMI	ICS_ENABLE_NMI
IC_PARITY_MASK	ICS_PARITY_MASK

IC_PARITY_OFFSET	ICS_PARITY_OFFSET
IC_SWNMI_MASK	ICS_SWNMI_MASK
IC_SW_NMI	ICS_SW_NMI
LO	ICS_LO
LOB	MPS_LOB
MAXMSGSZ	MPS_MAXMSGSZ
MAX_REG	ICS_MAX_REG
MAX_SLOT	ICS_MAX_SLOT
MB2_TPDT	MPS_MB2_TPDT
MG_BGDC	MPS_MG_BGDC
MG_BGLI	MPS_MG_BGLI
MG_BGLL	MPS_MG_BGLL
MG_BGOVHD	MPS_MG_BGOVHD
MG_BGRANT	MPS_MG_BGRANT
MG_BJLI	MPS_MG_BJLI
MG_BJOVHD	MPS_MG_BJOVHD
MG_BRDCST	MPS_MG_BRDCST
MG_BRDP	MPS_MG_BRDP
MG_BREJ	MPS_MG_BREJ
MG_BREQ	MPS_MG_BREQ
MG_BRLI	MPS_MG_BRLI
MG_BRML	MPS_MG_BRML
MG_BROVHD	MPS_MG_BROVHD
MG_BRPI	MPS_MG_BRPI
MG_BRSP	MPS_MG_BRSP
MG_BRTC	MPS_MG_BRTC
MG_BRTI	MPS_MG_BRTI
MG_BRUD	MPS_MG_BRUD
MG_DA	MPS_MG_DA
MG_DFBIND	MPS_MG_DFBIND
MG_DONE	MPS_MG_DONE
MG_DUTY1C	MPS_MG_DUTY1C
MG_DUTY2C	MPS_MG_DUTY2C
MG_ESOL	MPS_MG_ESOL
MG_MT	MPS_MG_MT
MG_NFDP	MPS_MG_NFDP
MG_NFFL	MPS_MG_NFFL
MG_NFOVHD	MPS_MG_NFOVHD
MG_NFPI	MPS_MG_NFPI
MG_NFSP	MPS_MG_NFSP
MG_NFTC	MPS_MG_NFTC
MG_NFTI	MPS_MG_NFTI
MG_REQ	MPS_MG_REQ
MG_RES	MPS_MG_RES
MG_RI	MPS_MG_RI
MG_RQMF	MPS_MG_RQMF
MG_RQNF	MPS_MG_RQNF
MG_RQSF	MPS_MG_RQSF
MG_RQXF	MPS_MG_RQXF

MG_RRMSK	MPS_MG_RRMSK
MG_RRTMSK	MPS_MG_RRTMSK
MG_RSCN	MPS_MG_RSCN
MG_RSET	MPS_MG_RSET
MG_RSNE	MPS_MG_RSNE
MG_SA	MPS_MG_SA
MG_TERR	MPS_MG_TERR
MG_UMDP	MPS_MG_UMDP
MG_UMOVHD	MPS_MG_UMOVHD
MG_UMPI	MPS_MG_UMPI
MG_UMSP	MPS_MG_UMSP
MG_UMTC	MPS_MG_UMTC
MG_UMTI	MPS_MG_UMTI
MG_UMUD	MPS_MG_UMUD
MG_UNSOL	MPS_MG_UNSOL
MY_SLOT_ID	ICS_MY_SLOT_ID
NRMPRIO	MPS_NRMPRIO
PARITY_CONTROL_TYPE	ICS_PARITY_ CONTROL_TYPE
PC16_CONFIG_OFFSET	ICS_PC16_ CONFIG_OFFSET
PC16_STATUS_OFFSET	ICS_PC16_ STATUS_OFFSET
PSB_CONTROL_TYPE	ICS_PSB_ CONTROL_TYPE
PSB_SLOT_ID_REG	ICS_PSB_ SLOT_ID_REG
READ_ICS	ICS_READ_ICS
SLOT_ID_OFFSET	ICS_SLOT_ID_OFFSET
SRLPRIO	MPS_SRLPRIO
WRITE_ICS	ICS_WRITE_ICS
agent_cmp	ics_agent_cmp
buf_count	mps_buf_count
close_chan	mps_close_chan
cpunum	ics_cpunum()
datbuf_t	mps_datbuf_t
db_join	dma_buf_join
dbuf_breakup	dma_buf_breakup
fetch_db	mps_fetch_db
fetch_mb	mps_fetch_mb
find_port	mps_find_port
find_transaction	mps_find_transaction
free_dbuf	mps_free_dmabuf
free_mbuf	mps_free_mbuf
free_msgbuf	mps_free_msgbuf
free_tid	mps_free_tid
get_dbuf	mps_get_dmabuf
get_mbuf	mps_get_mbuf
get_msgbuf	mps_get_msgbuf

get_soldata	mps_get_soldata
get_tid	mps_get_tid
get_unsoldata	mps_get_unsoldata
handle_err	mps_handle_err
init_dmacb	impc_init_dmacb
lhid	mps_lhid()
mk_bgrant	mps_mk_bgrant
mk_brdcst	mps_mk_brdcst
mk_breject	mps_mk_breject
mk_mb2socid	mps_mk_mb2socid
mk_mb2soctohid	mps_mk_mb2soctohid
mk_mb2soctopid	mps_mk_mb2soctopid
mk_snf	mps_mk_snf
mk_sol	mps_mk_sol
mk_solrply	mps_mk_solrply
mk_unsol	mps_mk_unsol
mk_unsolrply	mps_mk_unsolrply
mpc_copyright	impc_copyright
mpc_send	impc_send
mpc_start	impc_start
msg_comp	mps_msg_comp
msg_dispatch	mps_msg_dispatch
msg_getbrlen	mps_msg_getbrlen
msg_getdstmid	mps_msg_getdstmid
msg_getdstpid	mps_msg_getdstpid
msg_getfraglen	mps_msg_getfraglen
msg_getlsnid	mps_msg_getlsnid
msg_getmsgtyp	mps_msg_getmsgtyp
msg_getprotid	mps_msg_getprotid
msg_getreqid	mps_msg_getreqid
msg_getrid	mps_msg_getrid
msg_getsrcmid	mps_msg_getsrcmid
msg_getsrcpid	mps_msg_getsrcpid
msg_gettransctl	mps_msg_gettransctl
msg_gettrnsid	mps_msg_gettrnsid
msg_getudp	mps_msg_getudp
msg_iscancel	mps_msg_iscancel
msg_iscompletion	mps_msg_iscompletion
msg_iseot	mps_msg_iseot
msg_iserror	mps_msg_iserror
msg_isreq	mps_msg_isreq
msg_proc	mps_msg_proc
msg_que	mps_msg_que
msg_setbglen	mps_msg_setbglen
msg_setbrlen	mps_msg_setbrlen
msg_setcancel	mps_msg_setcancel
msg_setduty	mps_msg_setduty
msg_setrid	mps_msg_setrid
msg_setsrcpid	mps_msg_setsrcpid

msg_showmsg	mps_msg_showmsg
msgbuf_t	mps_msgbuf_t
myslotid	ics_myslotid()
open_chan	mps_open_chan
requestid	mps_requestid()
set_rid	mps_set_rid
setup_dmabufs	impc_setup_dmabufs
sic_cb	impc_sic_cb
slot_t	ics_slot_t
soc_cb	impc_soc_cb
sol_deque	impc_sol_deque
sol_que	impc_sol_que
transportcopyright	mps_copyright

SEE ALSO

System V/386 Device Driver Interface/Driver-Kernel Interface Reference Manual

NAME

`ttyswitch` – switches a line to a new host on the communications controller

SYNOPSIS

`/sbin/ttyswitch` *new_host_id*

DESCRIPTION

`ttyswitch` causes the issuing host to temporarily relinquish control of a line on the communications controller to another host. This program sends a message to the `atcs` server. If at the time the message is received by the server, and the new host has not also enabled the line, an error response is returned immediately.

Command line options are:

`new_host_ID`
: The host to which control is switched. Specify this option by using its MULTIBUS II iPSB slot-ID.

FILES

`/sbin/ttyswitch`

3. FUNCTIONS AND LIBRARIES

3. FUNCTIONS AND LIBRARIES

3 Functions And Libraries

Interconnect-Space Application Interface

This chapter lists the functions that make up the Interconnect-Space Application Interface. This interface is defined in the following routines:

- `ics_find_rec`
- `ics_read`
- `ics_write`

The functions take a file descriptor as the first argument. The file descriptor is obtained by opening the special device `/dev/ics` with the system call open. The routines are synchronous and return on completion of the request.

The interconnect space of any board on the MULTIBUS II backplane can be accessed from a System V/386 application using these functions. Accessing a board's interconnect space is useful for system maintenance and administrative functions.

NAME

`ics_find_rec` – finds a specific record in the interconnect space of a board

SYNOPSIS

`register ics_find_rec` (*fd, slot, recordid*)
`int` *fd*; `unsigned short` *slot*; `unsigned char` *recordid*; `int register;`

DESCRIPTION

This function is used to find a specific record in the interconnect space of a board. *slot* is the slot number of the board whose interconnect space is searched for a record with the record ID *recordid*. If such a record is found, its starting register number is returned. If the interconnect space register can not be found in the specified slot, a -2 is returned. Otherwise, -1 is returned.

FILES

`/usr/lib/libmb2.a`

NAME

`ics_read` – reads interconnect registers of the board and returns the values in the buffer

SYNOPSIS

`ret` = ics_read (*fd, slot, register, buf, count*); `int` *fd*; `unsigned short` *slot*; `unsigned short` *register*; `char` **buf*; `int` *count*; `int ret`;

DESCRIPTION

`ics_read` reads the interconnect registers of the board in the designated slot and returns the values in the buffer pointed to by *buf*. *count* number of registers are read starting from *register*.

If there is no board in the designated *slot*, or, if the register number specified does not exist in the interconnect space of the board, the read value is undefined. Thus, to determine if a board is present in a slot, the vendor ID registers in the interconnect space should be used. Zero (0) in the vendor ID register is defined to indicate the absence of a board.

FILES

`/usr/lib/libmb2.a`

NAME

`ics_write` – writes into interconnect registers of the board

SYNOPSIS

`ret = ics_write` (*fd, slot, register, buf, count*); `int` *fd*; `unsigned short` *slot*; `unsigned short` *register*; `char *`*buf*; `int` *count*; `int ret;`

DESCRIPTION

This function will write into interconnect registers of the board in *slot* number slot. *count* registers starting from *register* number register of the board are written from the values provided in the buffer pointed to by *buf*.

If there is no board in the designated *slot*, or, if the register number specified does not exist in the interconnect space of the board, the results are undefined.

Be careful when using `ics_write`. With `ics_write`, an applications program can reset any board on the bus, possibly resulting in a loss of data.

FILES

`/usr/lib/libmb2.a`

4. FILE FORMATS

4 Intel's System V/386 File Formats

Intel's System V/386 File Formats

This chapter lists Intel's System V/386 file formats. This chapter applies only to MULTIBUS II.

- `config`

NAME

config – MULTIBUS II configuration file format

DESCRIPTION

config is the bootserver's configuration file which gives the boot parameter string for hosts on MULTIBUS II and also specifies the load image that is returned by the bootserver(1iref).

Information for each host is grouped into a server part and a client part with the server part enclosed in square brackets ([ ]). Each of the two parts consists of "parameter/value" pairs that are separated by semicolons (;). Each parameter/value pair is made up of an alpha-numeric parameter name, an equal sign (=), and a value. These separate parts can be separated by white space which may be spaces, tabs, or end-of-lines (newlines). Comments can be inserted in white space as a hash mark (#) followed by the textual comment and terminated by an end-of-line.

Each entry is for a host specified by the parameter BL_host_ID in the server part of the entry. There is one special host entry that has the host number of GLOBAL. The GLOBAL entry has parameters that apply to all of the hosts. The parameter BL_second_stage specifies the filename of the default second stage bootloader for the host. If there is no default second stage bootloader specified, the bootloader specified in the GLOBAL host entry is given. If no GLOBAL second stage bootloader is specified in the configuration file, a default file is chosen by the bootserver. The boot parameter string given to a host is made up of the client part of that host's entry appended with the GLOBAL entry.

Replacement strings are parameter values that start with a dollar sign ($) and are followed by the name of a parameter that appears in the client portion of the GLOBAL entry. The dollar sign and the parameter are textually replaced with the value of the GLOBAL entry. This allows parameterized host entry specification.

The format of the file is:

```
<configfile> := <globalconfig> { <hostconfig> } ...
<globalconfig> := <ws> "[" <serverpart> <ws> "]"
    <consumerpart>
    <hostconfig> := <ws> "[" <serverpart> <ws> "]"
          <consumerpart>
<serverpart> := <parameter> { <ws> ";" <parameter> ... }
<consumerpart> := <parameter> { <ws> ";" <parameter> ... }

<parameter> := <ws> <parametername> <ws> "="
          <parametervalue>
<parametername> := <1to31alphanumericsor_>
<parametervalue> := <ws> <element> { <ws> <element> ... }
<element> := <string> | <number> | <substitution> |
          <quote> | <backslash>
<string> := <nonsyntaxcharacters>
<number> := <decimaldigits> [ "T" ] | <hexdigits> "H"
          | <octaldigits> "Q" | <binarydigits> "Y"
<substitution> := "$" <parametername>
```

```
 <quote> := """ <string> """ | "'" <string> "'"
<backslash> := "\" <anycharacter>
<ws> := <whitespace>
<whitespace> := <blank> | <tab> | <carriagereturn> |
          <linefeed> | <comment>
<comment> := "#" <characters> <linefeed>
```

Details on MULTIBUS II configuration are available in the MULTIBUSR II System Architecture Bootstrap Specification.

Here is a sample config file. Notice that the client part of the GLOBAL entry contains both parameters that are to be passed to each host in the boot parameter string and parameters that are used in replacement strings in the later host entries. There are host entries for two processor hosts that load a computing image; hosts 2 and 3 load and run the System V/386 operating system.

```
# #           MBII System Sample Configuration File
#

[BL_HOST_id = GLOBAL];
         PCI_host = 0;
         unix_host1 = 2;
         unix_host2 = 3;

# PCI Host
[BL_host_id = $PCI_host];
         BL_QI_Master = $unix_host1;
         BL_target_file = /stand/pci258;
         BL_mode = p;

# Unix Host #1
[BL_Host_id = $unix_host1;BL_Method = Quasi];
         BL_target_file = /stand/unix;

# Unix Host #2
[BL_Host_id = $unix_host2;BL_Method = Quasi];
         BL_target_file = /stand/unix;
```

NOTES

If the stand partition exists, any user-supplied configuration files must reside on the stand partition.
Parameter names are not case sensitive.
Parameter values are case sensitive based on context.
System V/386 filenames are case sensitive.

FILES

/stand/config

SEE ALSO

bootserver(1iref)

5. DEVICE INFORMATION

5. DEVICE INFORMATION

5 Device Information

Disk And Tape Device Drivers

This chapter lists information about System V/386 device drivers for MULTIBUS-based devices. The following types of device drivers are available on MULTIBUS-based systems:

- flexible disk device drivers
- hard disk device drivers
- tape device drivers

NAME

`fddd` – MULTIBUS flexible disk device drivers

DESCRIPTION

There are three MULTIBUS flexible disk device drivers:

i214
i224a
i258

Flexible disk drivers provide access to diskettes as block and character devices. Diskettes must be formatted before use (see `format`(1)). Both 5.25-inch and 3.50-inch formats are supported.

Flexible disk device file names are listed in the `/dev` directory. Each filename corresponds to a specific major device number, minor device number pair. The minor device number specifies the drive number, the format of the disk, and the partition number. The format of flexible disk device names is:

`/dev/ {r}dsk/f{0,1} {5h,5d9,5d8,5d4,5d16,5q,3h} t`

`rdsk`	selects the raw device interface
`dsk`	selects the block device interface
`0 or 1`	selects the drive to be accessed: f0 selects drive 0, f1 selects drive 1.
`5h`	indicates 5.25: high density (1.2Mbytes) diskette format
`5d9`	indicates 5.25: double density, 9 sectors per track (360 Kbytes) diskette format
`5d8`	indicates 5.25: double density, 8 sectors per track (320 Kbytes) diskette format flexible disk device drivers
`5d4`	indicates 5.25: double density, 4 sectors per track (360 Kbytes) diskette format
`5d16`	indicates 5.25: double density, 16 sectors per track (320 Kbytes) diskette format
`3h`	indicates 3.50: high density (1.44Mbytes) diskette format `t` indicates that the entire diskette will be accessed.

Besides the file naming convention just described, some of the diskette formats have alias names that correlate to previous releases. The following lists the aliases for the formats that have them:

format	**alias**
5h	q15d
5d8	d8d
5d9	d9d

For example, the device file `/dev/rdsk/f0q15dt` is equivalent to the device file `/dev/rdsk/f05ht`.
Following is a list of the supported `ioctl()` calls provided by MULTIBUS flexible disk device drivers.

`Get Parameters` (*V_GETPARMS*)
: This command gets configuration parameters for the current device and partition, and returns them to the user in the cmdarg structure. The disk_parms structure is defined in `/usr/include/sys/vtoc.h`.

`Format Track` (*V_FORMAT*)
: This command causes the specified track to be formatted. The format structure is defined in `/usr/include/sys/vtoc.h`.

NAME

hddd – MULTIBUS hard disk device drivers

DESCRIPTION

There are three MULTIBUS hard disk device drivers:

i214
i224a
i258

These drivers all require some similar disk information and all support the same set of `ioctl()` calls.

Hard disk device drivers require the Volume Label, Physical Description Information (`pdinfo`), and Volume Table of Contents (`vtoc`) information to access a given hard disk drive. To allow disks to be interchangeable, the information is stored on the disk itself. All of this information is placed on the device during system installation.

`Volume Label`	The driver reads the volume label to determine disk characteristics and to locate the `pdinfo`. The volume label is read when the driver first opens a device. For data structure definitions, see `ivlab` in the file `/usr/include/sys/ivlab.h`.
`pdinfo`	The `pdinfo` contains additional device characteristics, and pointers to the `vtoc` and alternate table. `pdinfo` is read when the driver first opens a device. For data structure definitions, see `pdinfo` in the file `/usr/include/sys/vtoc.h`.
`vtoc`	The `vtoc` contains the partition information required by the driver to access the hard disk drive. `vtoc` is read when the driver first opens a device. For data structure definitions, see `vtoc` in the file `/usr/include/sys/vtoc.h`.
`SW alt inf`	This is the software bad block handling table. It is in the file `/usr/include/sys/alttbl.h`.
`mdl`	Manufacturer's Defect List. This is the defective track list read from the hard disk drive. It is in the file `/usr/include/sys/bbh.h`.

Following is a list of the supported `ioctl()` calls provided by MULTIBUS hard disk device drivers.

`Get Parameters` (*V_GETPARMS*)
This command gets configuration parameters for the current device and partition, and returns them to the user in the `cmdarg` structure. The `disk_parms` structure is defined in `/usr/include/sys/vtoc.h`.

`Format Track` (*V_FORMAT*)
This command causes the specified track to be formatted. The format structure is defined in `/usr/include/sys/vtoc.h`.

`Configure Driver` (*V_CONFIG*)
: This command reconfigures the drive with the values passed in the `config` structure defined in `/usr/include/sys/vtoc.h`.

`Configure Driver` (*V_REMOUNT*)
: This command changes the device state flags so when the next open is issued for the device, configuration information is read from the disk and the driver's internal tables are reinitialized to reflect the new configuration. No partition other than zero can be open, and the user id must be zero to execute this command.

`Load Volume Label` (*V_LV_LAB*)
: This command loads the driver's internal volume label with the data in the `cmdarg` structure. No partitions other than zero can be open. The user id must be superuser. The structure is defined in `/usr/include/ivlab.h`.

`Upload Volume Label` (*V_U_VLAB*)
: This command loads the `user's cmdarg` structure with data from the driver's internal volume label. The structure is defined in `/usr/include/sys/ivlab.h`.

`Read Volume Label` (*V_R_VLAB*)
: This command loads the driver's internal volume label with the data from the hard disk. This command can only be used on partition zero, by user id zero, and the disk must contain a valid volume label. Refer to `/usr/include/sys/ivlab.h` for the starting byte offset of the volume label.

`Write Volume Label` (*V_W_VLAB*)
: This command writes the driver's internal volume label to the volume label in the hard disk. This command can only be used on partition zero, by user id zero, and the disk must have a properly formatted track zero.

`Load Defect Info` (*V_L_MDL*)
: This command adds the data in the `cmdarg` structure to the driver's internal defect list. Only partition zero may be open, and the command may be used only by user id zero.

`Upload Defect Info` (*V_U_MDL*)
: This command returns one entry from the driver's internal defect list to the `user's cmdarg` structure.

`Read Defect List` (*V_R_MDL*)
: This command loads the driver's internal defect list with the data from the hard disk. The driver must have a valid volume label in its tables. The disk must have a valid defect list on the next to last cylander for an ST-506 interface, or on the third from last cylander for an ESDI interface.

`Write Defect List` (*V_W_MDL*)
: This command writes the driver's internal defect list to the appropriate area of the hard disk. After the write is complete, the internal defect list is cleared to zero. Only partition zero

may be open, and the command may be used only by user id zero. The driver must have a valid volume label in its tables.

SEE ALSO

`i214`(7iref), `i224a`(7iref), `i258`(7iref)

NAME

tdd – MULTIBUS tape device drivers

DESCRIPTION

The formats for tape files are:

/dev/rmt/c0s0n	no rewind on close, no retension on open
dev/rmt/c0s0	rewind on close, no retension on open

These files refer to the QIC-24/QIC-02 basic cartridge tape streamer. Only raw character interface files are provided.

A standard tape consists of several 512-byte records, terminated by an end-of-file (EOF). To the extent possible, the system treats the tape like any other file. As in other raw devices, seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF so that the next read will return the next record.

A tape opened for reading or writing is assumed to be positioned as desired; that is, reading or writing occurs at the current position. It is possible to read and write multifile tapes by using the non-rewinding tape file.

When a rewind-on-close file (c0s0) is closed, the tape is rewound. Also, if the file was opened for writing and data was written, a double EOF (a double tape mark) is written when the file is closed.

When a no-rewind-on-close file that was opened for writing is closed, if data was written, one EOF is written and the tape is positioned after the EOF. When a no-rewind-on-close file is closed after being opened for read-only, the tape is positioned after the EOF following the just-read data.

The following ioctls are supported:

T_RETENSION	retension the tape
T_RWD	rewind the tape to Beginning of Tape (BOT)
T_LOAD	rewind the tape to BOT
T_UNLOAD	rewind the tape to BOT
T_ERASE	erase the tape and leave it at BOT
T_WRFILEM	write an EOF (tape mark)
T_RST	reset the tape device
T_SFF	skip forward arg files

When a T_RWD, T_RETENSION, T_LOAD, or T_UNLOAD ioctl is requested after a write operation, a double EOF (double tape mark) is written before the ioctl is executed.

FILES

/dev/rmt/c0s0n
/dev/rmt/c0s0

SEE ALSO

i214tp(7iref), i224atp(7iref), i258tp(7iref)

6. DEVICE DRIVERS

6. DEVICE DRIVERS

6 Device Drivers

Device Drivers

This chapter lists System V/386 device drivers for MULTIBUS-based devices. The following drivers support both MULTIBUS I and MULTIBUS II systems:

- console
- iasy
- ramd
- rtc

The following drivers support MULTIBUS I systems:

- i214
- i214tp
- i546
- i8251

The following drivers support MULTIBUS II systems:

- atcs
- bpg
- cci
- d258
- dma
- edlina
- i224a
- i224atp
- i258
- i258tp
- i354
- i410
- i530
- ics
- mpc
- mps
- ots
- rci

NAME

console – console port device driver

DESCRIPTION

The console device driver provides an interface to the device driver controlling the physical port assigned as the console port (for example, the i354 device driver). Operations performed to the console are redirected by the console device driver to the device driver controlling the physical device for interpretation and handling.

Following is a list of device drivers that can control physical ports in MULTIBUS systems. The console driver provides an interface to one of these device drivers:

```
i354
i8251
rci
```

If you are writing a device driver in order for it to become the console, it must call consregister with the following parameters:

```
void consregister(ci,co,dev)
int (*ci)();
int (*co)();
dev_t dev;
```

The *co* and *ci* fields contain pointers to the routines which handle the output and input operations, respectively, for the kernel (debug console). The *co_dev* field contains the major-minor number value for the device driver controlling the /dev/console. These fields are set automatically if the i354, i8251,or rci devices are used as the console. See the System V/386 Device *Driver Interface Specification* for information on writing a device driver.

FILES

/etc/conf/cf.d/mdevice	mdevice entries
/etc/conf/sdevice.d/console	sdevice entries
/etc/conf/pack.d/console/Driver.o	console device driver object module
/usr/include/sys/conf.h	console driver specific definitions

SEE ALSO

ioctl(3), i354(7iref), i8251(7iref), iasy(7iref),
rci(7iref), termio(7) *System V/386 Device Driver Interface/Driver-Kernel Interface Reference Manual*

NAME

`iasy` – asynchronous (terminal) device driver

DESCRIPTION

The `iasy` driver handles the interface between MULTIBUS terminal device drivers and applications as described by the `termio`(7) specification.

To configure the `iasy` driver, modify the following variables in the `/etc/conf/pack.d/iasy/space.c` file:

iasy_num Contains the number of terminal devices to be supported by the system. This number must correspond to the size of the `iasy_tty` and `iasy_hw` arrays.

The `iasy` driver also supports the `termio`(7) ioctls for use by applications.

FILES

`/usr/include/sys/iasy.h`	definitions for data structures
`/etc/conf/pack.d/iasy/space.c`	configuration information

SEE ALSO

`termio`(7), `atcs`(7iref), `i354`(7iref), `i546`(7iref), `i8251`(7iref) `ics`(7iref)

NAME

ramd – MULTIBUS RAM disk driver

DESCRIPTION

The RAM disk driver provides block device access to system memory. It can be used to set up a RAM-based file system. The memory for the RAM disk is mapped from system memory. The *roving: RAM disk is built in sysseg out of user memory. Memory is defined by a structure that gives the starting address and size of the memory to be managed. The structure is called ramdinfo and is defined in the file /usr/include/sys/ramd.h. The format of the structure is:

```
struct ramd_info {

        ulong   ramd_size;       /* Size of disk in bytes */
        ulong   ramd_flag;       /* State flags; see defs below */
        dev_t   ramd_maj;        /* Major device for load */
        dev_t   ramd_min;        /* Minor device for load */
        ulong   ramd_state       /* Runtime state */
        caddr_t ramd_paddr;      /* Kernel virtual address */
  };
```

Flag definitions:

```
#define RAMD_STATIC      0x01    /* RAM Disk statically allocated */
#define RAMD_RUNTIME     0x02    /* Runtime definable RAM Disk */
#define RAMD_LOAD        0x04    /* Auto fill RAM disk at init */
```

If the RAMD_RUNTIME flag is chosen, the interface to the RAM disk driver functions is through four ioctl calls. A utility must be written that uses them to allocate and free up space. The ioctl calls are:

RAMD_IOC_GET_INFO	returns size, flags, and base
RAMD_IOC_R_ALLOC	defines a roving ram disk
RAMD_IOC_R_FREE	frees a roving ram disk
RAMD_IOC_LOAD	loads a ramd partition

The ioclt calls use the ramd_info structure, described above. The ioctl calls are defined in the file /usr/include/sys/ramd.h. The following is a list of error conditions that may be returned.

`No Device or Address [ ENXIO ]`
: If the device specified is unknown, this message is printed.

`Not Enough Core [ ENOMEM ]`
: If the ram disk cannot be set up as specified, the memory is not available and this message is printed.

`Invalid Argument [ EINVAL ]`
: This message is printed if a bad ioctl request is specified or the ram disk size specified is zero.

NOTES

When configuring RAM disks, make sure the size of the RAM disk is specified in bytes and not blocks. Also, the RAM disk size and starting address must be a multiple of 4Kb (which is the page size).

FILES

`/usr/include/sys/ramd.h`	Driver specific definitions
`/etc/conf/pack.d/ramd/space.c`	Driver specific configuration

NAME

rtc – MULTIBUS clock driver

DESCRIPTION

The rtc driver provides a multiplexed interface to the iSBC 546 clock on MULTIBUS I systems and the CSM clock on MULTIBUS II systems. It conforms to the AT386 real time clock interface thus providing a consistent interface to the System V/386 kernel.

The interaction is based on a client/server model. The i546 and csmclk drivers are the servers that handle the read/write clock functions; the clock driver is the client which converts the server clock formats to standard real time format and provides the kernel and user application external interfaces. The csmclk driver supports both the CSM001 and CSM002 boards.

The kernel interface to the rtc driver functions is through two routines that get and set the time using an address to a structure called rtc_t. The two requests are clkget and clkput. These routines fail and return a -1 if any hardware error occurs while reading or writing the rtc. The structure is defined in the file /usr/include/sys/rtc.h. The format of the structure is:

```
struct  rtc_t  {
  unsigned char  rtc_sec;        /* second of minute, 0-59 */
  unsigned char  rtc_asec;       /* second alarm */
  unsigned char  rtc_min;        /* minute of hour, 0-59 */
  unsigned char  rtc_amin;       /* minute alarm */
  unsigned char  rtc_hr;         /* hour of the day, 1-23 */
  unsigned char  rtc_ahr;        /* hour alarm */
  unsigned char  rtc_dow;        /* day of the week, 1-7 */
  unsigned char  rtc_dom;        /* day of the month, 1-31 */
  unsigned char  rtc_mon;        /* month of the year, 1-12 */
  unsigned char  rtc_yr;         /* year of the century, 0-99 */
  unsigned char  rtc_statusa;    /* status register A */
  unsigned char  rtc_statusb;    /* status register B */
  unsigned char  rtc_statusc;    /* status register C */
  unsigned char  rtc_statusd;    /* status register D */ };
```

The user application interface to the rtc driver functions is through two ioctl functions that are the same as the AT386 clock ioctl functions. The two requests are RTCRTIME and RTCSTIME. They get and set the time using an address to the same structure called rtc_t shown above. The ioctls are defined in the file /usr/include/sys/rtc.h.

The following is a list of error conditions that may be returned.

I/O Error [EIO]
: This error is printed if the driver gets an error while trying to read the real-time clock chip.

Permission Denied [EACCESS]
: This message is printed if the user trying to access the clock device (to set the time) is not the superuser. The message is also printed if the clock could not be set.

`Invalid Argument [ EINVAL ]`
This message is printed if one of the fields used to set the time is outside the prescribed limits.

FILES

`/usr/include/sys/clockcal.h`	Driver specific definitions
`/usr/include/sys/rtc.h`	Real time clock specific definitions

SEE ALSO

rtc(7), i546(7iref)

NAME

`i214` – iSBCR 214 peripheral controller disk device driver

DESCRIPTION

The `i214` device driver provides an interface to peripheral devices controlled by an Intel iSBC 214 peripheral controller board or an iSBC 221 peripheral controller board. These devices can be hard disk drives and floppy drives. (For cartridge tape support, see `i214tp`(7iref)) ST506-interface hard disk drives are supported. This driver supports up to four hard disk drives and four floppy drives (360 KByte capacity).

The interface to the `i214` driver is the standard System V/386 block device interface.

CONFIGURATION

Configuration of the `i214` driver is defined by the following data structures:

```
minor table
board table
drive table
partition table
```

These tables are in the file `/etc/conf/pack.d/i214/space.c` . Users can change the entries the minor table and board table to configure the driver. Changing the contents of the drive and partition tables is not recommended.

The configuration for the `i214tp`(7iref) driver is handled by the same data structures as the `i214` driver. The `i214` driver must be configured into the system in order to use the `i214tp`(7iref) driver.

Minor Table

The minor table maps a logical device's minor number to a partition on a physical device. A physical device consists of a specific unit on a board. For more information on minor number assignments, see `i214minor` in the file `/etc/conf/pack.d/i214/space.c`.

Minor Table	Unit	Board
0 - 15	Wini0	0
16 - 31	Wini1	0
32 - 47	Wini2	0
48 - 63	Wini3	0
64 - 78	Floppy0	0
79 - 93	Floppy1	0
94 - 108	Floppy2	0
109 - 123	Floppy3	0
124,125	Tape0	0
126,127	Tape1	0
128 - 143	Wini0	1
144 - 159	Wini1	1
160 - 175	Wini2	1
176 - 191	Wini3	1
192 - 206	Floppy0	1
207 - 221	Floppy1	1
222 - 236	Floppy2	1
237 - 251	Floppy3	1
252,253	Tape0	1
254,255	Tape1	1

Board Table

The board table is an array of structures containing the information required to access each board in the system, and each unit on the board. The minor number from the `minor table` indicates a board in the board table. To configure the `i214` driver, find the `i214cfg` structure in the file `/etc/conf/pack.d/i214/space.c`. There are three configurable values in each entry of this structure:

c_wau	wakeup address
c_devcod	devices
c_level	interrupt level

These values begin the first line of of each entry, in the format:

`c_wua, i214TYPE, FLPYTYPE, TAPETYPE, c_level`

Note that there is more information in each structure entry, but it is not configurable. If you are putting a new entry into the board table, use the wakeup address (`c_wua`) and interrupt level (`c_level`) that were set in hardware when the board is installed. If either of these two values is wrong, the driver will not be able to find the board.

The device (c_devcod) literals, `i214TYPE`, `FLPYTYPE`, and `TAPETYPE`, are defined earlier in the `i214/space.c` file to correspond to devices listed in the `/usr/include/sys/i214.h` file. The list of devices in the `i214.h` file contains all valid devices for the `i214` driver.

You can change the `FLPYTYPE` literal definition from `DEV5FLPY` (low-density 5.25: diskette drive) to `DEV8FLPY` (high-density 5.25: diskette drive) if the board is an `i221` board. Otherwise, do not change the definitions of the c_devcod literals.

The remainder of each entry in the `board table` is a list of pointers to the drive table for each device connected to the board. Do not modify this list. If you are adding a board, copy the list from an old board entry to the new entry.

The drive table and partition table contain lists of the possible configurations for the driver. If you are configuring the driver, do not modify the contents of either the drive table or the partition table.

Hard Disk Information

This device driver requires certain information about a disk drive in order to access the drive. This information is stored on the disks themselves. *Hard Disk Device Drivers (5iref)* discusses the disk drive information that this and other disk device drivers need. Also, certain `ioctl()` calls are supported by all MULTIBUS disk device drivers. These calls are listed in the hard disk device drivers (5iref) man page.

Flexible Disk Information

There is a device file in the /dev directory for each type of valid flexible disk drive. The device filenames follow a naming convention that is discussed in the flexible disk device drivers(5iref) man page.

FILES

`/usr/include/sys/i214.h`
`/etc/conf/pack.d/i214/space.c`

SEE ALSO

Flexible disk device drivers (5iref) Hard disk device drivers (5iref) `i224a`(7iref), `i258`(7iref)

NAME

i214tp – iSBC 214 peripheral controller tape device driver

DESCRIPTION

The **i214tp** device driver provides an interface to tape devices controlled by an iSBC 214 peripheral controller board. The driver supports 1/4-inch QIC-02 cartridge magnetic tape only.

The configuration for the **i214tp** driver is handled by the same data structures as the **i214** driver. The **i214** driver must be configured into the system in order to configure the i214tp driver.

There is a device file in the **/dev** directory for each type of valid tape drive. The device filenames follow a naming convention that is discussed in the tape device **drivers**(5iref) man page.

SEE ALSO

drivers(5iref), **i214**(7iref)

NAME

i546 – iSBC 546 multi-port serial controller

DESCRIPTION

The i546 device driver provides a termio interface to the iSBC 546 or iSBC 549 serial controller card on MULTIBUS I. Because of compatible hardware interfaces, it also interfaces with the iSBC 547 and iSBC 548.

The i546 driver is configured by information in structures initialized in the space.c file. The maximum number of 546 boards (including 547, 548, and 549 boards) that will be initialized is set by the variable NUM546 (default is 2). The MULTIBUS I memory address, I/O port, and interrupt level of each board are specified in an array of i546cfg structures. The structure is defined as follows:

```
struct i546cfg {
        long  c_addr;        /* board's physical address */
        int   c_port;        /* board's interrupt address */
        int   c_level;       /* board's interrupt level */
        }
```

c_addr gives the MULTIBUS I physical address of the first byte of the board's memory region. c_port specifies the MULTIBUS I I/O port that interrupts the board. c_level specifies the interrupt level with which the board will interrupt the CPU. Refer to the *iSBCR 546/547/548 High Performance Terminal Controllers Hardware Reference Manual* for default settings.

FILES

/etc/conf/pack.d/i546/Driver.o	i546 driver object module
/usr/include/sys/i546.h	i546 specific definitions
/usr/include/sys/clockcal.h	clock/calendar definitions
/etc/conf/cf.d/mdevice	mdevice entries
/etc/conf/sdevice.d/i546	sdevice entries
/etc/conf/pack.d/i546/space.c	i546 specific declarations and initializations

SEE ALSO

ioctl(3), termio(7)

NAME

`i8251` – console terminal/iSBXT 351 driver

DESCRIPTION

The `i8251` device driver, together with the `iasy`(7iref) driver, provides a termio interface to the serial port of an iSBC 386/xx or an iSBX 351 module attached to an iSBC 386/xx board. It also provides kernel debug port services via the `console`(7iref) interface.

Configure the i8251 device driver using the sdevice (4) file format.

FILES

`/etc/conf/pack.d/i8251/Driver.o`	`i8251` driver object module
`/usr/include/sys/i8251.h`	`i8251` specific definitions
`/etc/conf/cf.d/mdevice`	mdevice entries
`/etc/conf/sdevice.d/i8251`	sdevice entries
`/etc/conf/pack.d/i8251/space.c`	`i8251` specific declarations and initializations

SEE ALSO

`sdevice`(4), `iasy`(7iref), `console`(7iref)

NAME

atcs – asynchronous terminal controller script device driver

DESCRIPTION

The **atcs** device driver, together with the **iasy**(7iref) driver, provides a termio interface to terminal controller scripts running on serial port controller boards within MULTIBUS II systems. The **atcs** device driver provides functions which implement the asynchronous terminal protocol.

The **atcs** device driver uses the MULTIBUS II transport message passing mechanism to communicate with servers running on serial controller boards (the iSBC 186/410 Serial Communications Controller board, for example) in the MULTIBUS II system.

The **atcs** device driver relies upon the **cci** device driver to set up communication paths to a server running on a serial controller board. Once the **cci** device driver has set up a channel for an **atcs** line, the **atcs** device driver may use that channel to communicate with a server controlling a port on the serial controller board.

The **/etc/conf/pack.d/ats/space.c** file contains the **atcs_info** structure. This structure holds information that maps slots and ports to minor numbers. This information specifies which ATCS lines in a system should be configured for System V/386 Release 4.0.

The **atcs_info** structure contains the following information:

slot	The board on which an **atcs** server may be running.
fmin	The first ATCS minor number to be allocated.
lmin	The last ATCS minor number to be allocated.
port	The first hardware port to be allocated.

Users enter appropriate values into the fields, and in this manner map ranges of minor number onto physical boards and ports.

The mapping of slot/port to minor number can also be specified with the boot parameter string (BPS). If there is **atcs** information in the BPS, the system does not check to see if there is information in the **atcs_info** structure of the **space.c** file.

FILES

/etc/conf/cf.d/mdevice	mdevice entries
/etc/conf/sdevice.d/atcs	sdevice entries
/etc/conf/pack.d/atcs/Driver.o	atcs device driver object module
/etc/conf/pack.d/atcs/space.c	atcs specific declarations and initialization
/usr/include/sys/atcs.h	atcs specific definitions
/usr/include/sys/atcsmp.h	atcs message passing protocol definitions

SEE ALSO

cci(7iref), **iasy**(7iref), **termio**(7), **cci**(7iref), **bps**(7iref)

NAME

bps – Bootstrap Parameter String driver

DESCRIPTION

The **bps** driver provides an interface to the bootstrap parameter string (BPS) for the kernel, MULTIBUS II device drivers, and applications. When the operating system is loaded on a host, it uses the BPS to configure the system. For UNIX, the individual device drivers configure their tables and devices based on the information in the BPS.

The BPS consists of several entries of the form `parameter`=*value* that are separated by semicolons. The parameter must be unique within a parameter string for a particular host, so multiple entries can be specified by similar parameter names that can be matched with wildcards. The value can have multiple configuration parameters of the form `config`:*value* that are separated by commas. For example, the BPS entry for an asynchronous I/O device at host id 5 that is assigned to minor numbers 23 through 30 is a `ASYNC=hid:5,port:0-7,minor:23-30;`

CONFIGURATION

The **bps** driver configuration file contains the following configurable variables:

`bps_use_native`
: When this flag is set to 1, the **bps** driver uses the default BPS left in RAM by the MULTIBUS II Systems Architecture (MSA) firmware BPS manager. The BPS contains parameters from ROM, operator intervention, and the bootserver configuration file.

`bps_tokenized_value`
: In R1.0 of the MSA firmware, the BPS manager splits the value of the parameter into tokens. When this flag is set to 1, the **bps** driver parses through the value of the parameter.

`bps_testing`
: When this variable is set to 1, it lets an application use the `BPSINIT` command of the `ioctl` function, which allows specification of a bootstrap parameter string.

`bps_ram_addr`
: This is the physical address in memory where the MSA firmware BPS manager stores the bootstrap parameter string. This value is used by the **bps** driver in conjunction with `bps_testing` to access the BPS in systems which do not have the updated second-stage bootstrap loader.

BPS DRIVER KERNEL INTERFACE

The following routines define the **bps** driver Kernel Interface. The routines are synchronous and return on completion of the request. They assume that the character string is null-terminated. All routines return a 0 if successfully completed and a -1 or error number if there is a failure. Errors are listed later in this section.

bps_open
e_code = bps_open();

The **bps_open** function ensures that the **bps** driver is initialized correctly. This routine must be called by all device drivers which access the bps. It eliminates any dependency on the order in which the kernel calls the driver init routines.

```
bps_init
e_code = bps_init(string_p);
char string_p;
```

The bps_init function allows the caller to specify an alternate BPS. The bps driver assumes that the string is null-terminated, well-formed, and conforms to the syntax of the Bootstrap Parameter String. After this call returns, the bps driver uses this string to retrieve parameters or values.

```
bps_get_val
e_code = bps_get_val(string_p, vbuf_len, valbuf_p);
char string_p;
int vbuf_len;
char valbuf_p;
```

The bps_get_val function retrieves the value of the parameter pointed to by string_p and returns it in the buffer pointed to by valbuf_p. vbuf_len is the length of the buffer, so the contents of the buffer are valid only if the length of the returned character string is less than vbuf_len.

```
bps_get_wcval
e_code = bps_get_wcval(string_p, state_p, vbuf_len, valbuf_p);
char string_p;
int state_p;
int vbuf_len;
char valbuf_p;
```

The bps_get_wcval function retrieves the value of the parameter pointed to by string_p and returns the value in the buffer pointed to by valbuf_p. vbuf_len is the length of the buffer, so the contents of the buffer are valid only if the length of the returned character string is less than vbuf_len.

Simplistic wildcards are supported. The wildcard characters and "?" must always be the last character of the parameter name and the parameter name must have at least one other character besides the wildcard character. The "?" will match one character of the parameter name, while the "*" will match a sequence of characters. For example, the wildcards "async?" or "asy*" can be used to retrieve the following parameters:

```
async1 = major:4,minor:0-5,protocol:atcs;
async2 = major:4,minor:7-11,protocol:atcs;
```

Prior to the first call to bps_get_wcval, the value pointed to by state_p should be initialized to 0, and in subsequent calls to bps_get_wcval it should not be modified. The bps driver uses state_p as an index of successfully matched parameters.

```
bps_get_opt
e_code = bps_get_opt(valbuf_p, state2_p, string_p, config_code,
value_len, value_p);
char valbuf_p;
int state2_p;
char string_p;
int config_code;
int value_len;
char value_p;
```

The bps_get_opt function retrieves the value of the keywords pointed to by the string string_p and returns the value in the buffer pointed to by value_p. value_len is the length of the buffer, so the contents of the buffer are valid only if the length of the returned character string is less than value_len. valbuf_p is the value to scan and is the output of a prior call to either bps_get_val or bps_get_wcval. The value pointed to by state2_p should be initialized to 0, and in subsequent invocations.

```
bps_get_socket
e_code = bps_get_socket(value_p, port_p, hostid_p);
char  value_p;
int   port_p;
int   hostid_p;
```

The bps_get_socket function returns the host id and port id as long integers from the character string pointed to by value_p. This character string is the output from a previous call to bps_get_val, bps_get_wcval, or bps_get_opt.

```
bps_get_range
e_code = bps_get_range(value_p, lo_range_p, hi_range_p);
char value_p ;
int lo_range_p; int hi_range_p;
```

The bps_get_range function returns the lower and upper bounds of a range as long integers from the character string pointed to by value_p. This character string is the output from a previous call to bps_get_val, bps_get_wcval, or bps_get_opt.

```
bps_get_integer
e_code = bps_get_integer(value_p, int_p)
char  value_p;
int   int_p;
```

The bps_get_integer function returns the value of the character string pointed to by value_p as a long integer. The character string is the output from a previous call to bps_get_val, bps_get_wcval, or bps_get_opt. Overflow conditions are ignored.

BPS DRIVER LIBRARY INTERFACE

The following routines define the bps driver Library Interface. The routines are synchronous and return on completion of the request. They assume that the character string is null-terminated. All routines return a 0 if successfully completed and a -1 or error number if there is a failure. Errors are listed later in this section.

```
bpsopen
e_code = bpsopen();
```

The bpsopen function opens the bps device driver. It must be called by all applications which need access to the bps.

```
bpsclose
e_code = bpsclose();
```

The bpsclose function closes the bps device driver. It must be called by all applications when they have finished accessing the bps.

```
bpsinit
e_code = bps_init(Newbps);
char Newbps;
```

The bpsinit function allows the application to specify an alternate BPS for use. This function is provided for testing purposes only, so it will work if the BPS is configured for testing. The bps driver assumes that the string is null-terminated, well-formed, and conforms to the syntax of the Bootstrap Parameter String. After this call returns, the bps driver uses this string for references when retrieving parameters or values. The following functions have the same syntax as in the Kernel Interface: bps_get_val, bps_get_wcval, bps_get_opt, bps_get_integer, bps_get_range, and bps_get_socket.

SUPPORTED FEATURES

The bps driver supports the following ioctl() calls for use by applications:

BPSINIT This command requires that the bps driver be configured with the bps_testing flag enabled and that the caller's id is zero. If the argument addr is NULL, the bps driver reinitializes its internal data structures with the BPS in RAM. If the argument addr is a pointer to structure bps_ioctl, the bps driver reinitializes its internal data structures with the BPS pointed to by bps_ioctl->string_p. The bps driver assumes that the string is well-formed and conforms to the syntax specification of the BPS.

BPSGETPV The argument is a pointer to structure bps_ioctl. It returns a character string at the location pointed to by bps_ioctl->valbuf_p, which is the value of the parameter pointed to by bps_ioctl->string_p.

BPSGETWCPV
The argument is a pointer to structure bps_ioctl. It returns a character string at the location pointed to by bps_ioctl->valbuf_p, which is the value of the wildcard parameter pointed to by bps_ioctl->string_p.

BPSGETOPTS
The argument is a pointer to structure bps_ioctl. It returns an integer value at the location pointed to by bps_ioctl->config_code, which is an index in the list of names pointed by

bps_ioctl->string_p. It also returns a character string pointed to by bps_ioctl->value_p, which is the value of the name. bps_ioctl->valbuf_p is the pointer to the character string; it is the value of a parameter name retrieved by either BPSGETPV or BPSGETWCPV.

BPSGETINTEGER
: This command converts a character string to an integer. The argument is a pointer to structure bps_ioctl. It returns an integer (at the location pointed to by bps_ioctl->lo_return_p) for the character string pointed to by bps_ioctl->value_p.

BPSGETSOCKET
: This command returns the host id and port id components. The argument is a pointer to structure bps_ioctl. It returns integer values (at the location pointed to by bps_ioctl->lo_return_p and bps_ioctl->hi_return_p) for the character string pointed to by bps_ioctl->value_p.

BPSGETRANGE
: This command returns the lower and upper bounds of a range. The argument is a pointer to structure bps_ioctl. It returns integer values (at the location pointed to by bps_ioctl->lo_return_p and bps_ioctl->hi_return_p) for the character string pointed to by bps_ioctl->value_p.

EXAMPLES

An example of how to use the bps routines is as follows:

```
if !(bpsopen()) {
     BPSstate1 = 0;
     while   !(bps_get_wcval("ASYNC",   &BPSstate1,   BPSvalbuflen,
&BPSvalbuf1) {
          BPSstate2 = 0;
          while !(bps_get_opt(&BPSvalbuf, &BPSstate2,
          "hid:minor:minor", &ConfigCode, &ValPntrLen, &ValPntr))
{
               switch (ConfigCode) {
                    1)
                         if !(bps_get_integer(ValPntr, &ThisHID))
                              dev_printf("illegal HID value0);
                         break;
                    2)
                    3)
                         ...
               }
          }
     }
}
```

The next example illustrates the format of the BPS for the mapping of host/port to minor numbers shown in the following table.

Host id	Port	Minor Number
6	3	5
6	4	6
6	5	7
7	0	8

```
ASYNC1=hid:6,port:3,minor:5;
ASYNC2=hid:6,port:4,minor:6;
ASYNC3=hid:6,port:5,minor:7;
ASYNC4=hid:7,port:0,minor:8;
```

If the driver can pack values, then ranges could be specified in the above mappings as follows:

```
ASYNC1=hid:6,port:3-5,minor:5-7; ASYNC2=hid:7,port:0,minor:8;
```

ERROR CODES

Possible error codes are as follows:

EFAULT	A memory address used in an `ioctl` is not a valid data address.
EINVAL	An invalid `ioctl` request is attempted or the parameters to the `ioctl` request are inconsistent.
EPERM	`BPSINIT` `ioctl` request is attempted with non-zero user id.
EBUSY	`BPSINIT` `ioctl` request is attempted while another `BPSINIT` is in progress or the **bps** is configured with the `bps_testing` flag disabled.
ENOMEM	The **bps** could not allocate enough memory to copy user parameters.
ENODEV	An `ioctl` request is made prior to the **bps** driver initializing its internal state.

FILES

`/usr/include/sys/bps.h`	Definitions for data structures.
`/etc/conf/pack.d/bps/space.c`	BPS driver configuration file.

SEE ALSO

MULTIBUS II System Architecture Bootstrap Specification

NAME

cci – communications controller interface device driver

DESCRIPTION

The cci device driver provides an interface to servers on serial port controller boards within a MULTIBUS II system. These servers manage the use of the serial ports on the controller boards.

The cci device driver functions encapsulate the Communications Controller Interface (CCI) protocol for use by device drivers within the system which implement a terminal controller interface. The MULTIBUS II Transport message passing mechanism is used by the cci device driver to communicate with a CCI server running on a serial controller board.

A terminal controller interface driver such as the **atcs** device driver relies on the cci driver to provide management of serial ports on a serial controller board. The cci driver handles the communication between the client host and the CCI server for serial port connection, dissolution, and switching. The **atcs** driver may request to attach, detach, or switch from, a channel to a serial port. These requests are made via the cci device driver.

FILES

`/etc/conf/cf.d/mdevice`	mdevice entries
`/etc/conf/sdevice.d/cci`	sdevice entries
`/etc/conf/pack.d/cci/Driver.o`	cci device driver object module
`/usr/include/sys/cci.h`	cci specific definitions
`/usr/include/sys/ccimp.h`	cci message passing protocol definitions

SEE ALSO

ioctl(3), **atcs**(7iref), termio(7)

NAME

`d258` – i82258 ADMA device driver

DESCRIPTION

The `d258` driver handles the internal low level interface between the `dma`(7iref) interface and the actual DMA hardware of a MULTIBUS II board.

To configure the `d258` driver, modify the following variables in the `/etc/conf/pack.d/d258/space.c` file:

d258_base	The base I/O port address for the i82258 chip.
d258_gmr	The bit mask to be loaded into the mode register of the i82258 chip. Do not modify this variable.
d258_gbr	The maximum number of contiguous bus cycles to be used for DMA. A zero is used to indicate unlimited length transfers are permitted.
d258_gdr	The number of bus cycles to wait between DMA bursts.
d258_sbx_base	The base address for DMA to/from the iSBX module.
*d258_sbx_sw_*	A flag indicating that dma has been configured
chan0_swconf	For iSBX channel 0.
chan1_swconf	For iSBX channel 1.

There are no ioctls for the `d258` driver. The `d258` driver is used to support the DMA interface required by the System V/386 Device Driver Interface specification, which is used in turn to implement other drivers which do have ioctl interfaces.

FILES

`/etc/conf/pack.d/d258/space.c` Configuration information

SEE ALSO

`dma`(7iref)
SystemV/386 Device Driver Interface/Driver-Kernel Interface Reference Manual

NOTES

DMA to or from iSBX connectors is not yet supported.

NAME

dma – DMA device driver

DESCRIPTION

The dma driver implements the internal DMA interface for device drivers as described in the *SystemV/386 Device Driver Interface/Driver-Kernel Interface Reference Manual.*

There are no configurable variables for the dma driver.

There are no ioctls for the dma driver. The dma driver is used to support the DMA interface required by the *System V/386 Device Driver Interface* specification, which is used in turn to implement other drivers that have ioctl interfaces.

SEE ALSO

i258(7iref)
SystemV/386 Device Driver Interface/Driver-Kernel Interface Reference Manual

NAME

edlina – External Data Link driver for iNA961 Release 3.0.

DESCRIPTION

The edlina driver provides a STREAMS interface that integrates with the System V/386 TCP/IP protocol stack. The edlina driver is the end of the streams queue pair providing the interface to the underlying External Data Link (EDL) offered by iNA961 Release 3.0. The Link Level Interface is provided by the edlina driver by transmitting and receiving Data Link packets using the communication interface provided by the i530 driver.

Configuration and tunable parameters for the edlina driver are defined in /etc/conf/pack.d/edlina/space.c. In order to change the driver configuration, this file must be edited and the kernel regenerated. These parameters are described below:

N_BOARDS	Number of 186/530 boards in the system. The default is 1.
EDL_MAX_	Maximum number of buffers posted for.
BUFS_POSTED	receiving data. The default is 16.

FILES

/etc/conf/pack.d/edlina/space.c	edlina driver configuration file
/etc/conf/pack.d/edlina/Driver.o	edlina driver object file
/etc/conf/node.d/edlina	edlina driver device node definition
/etc/conf/sdevice.d/edlina	edlina driver system device entry
/usr/include/sys/edlina.h	edlina driver user data structures
/etc/strcf	TCP/IP configuration file

SEE ALSO

i530(7iref)

NOTES

The following line will need to be added to the boot function in the /etc/strcf file to configure TCP/IP to use the edlina driver:

```
cenet ip /dev/edlina emd 0
```

There should be only one cenet (or senet) executable line in this file.

NAME

i224a – iSBC 186/224A peripheral controller disk device driver

DESCRIPTION

The i224a device driver provides an interface to peripheral devices controlled by a Peripheral Controller Interface (PCI) running on an Intel iSBC 186/224A peripheral controller board. These devices can only be hard disk drives and floppy drives. (For cartridge tape support, see i224atp(7iref).) ST506-interface hard disk drives are supported. This driver supports up to four hard disk drives and four floppy drives (360 KByte capacity). The driver supports one or two controller boards, with two boards as the default configuration. For more information on the PCI Interface, see the *iSBCR 186/224A Peripheral Controller Board User's Guide.*

The interface to the i224a driver is the standard System V/386 block and character device interface.

CONFIGURATION

The configuration for the i224a driver is handled by the following data structures: the minor table, the board table, the drive table, and the partition table. Users can change the entries the minor table and board table to configure the driver. Changing the contents of the drive and partition tables is not recommended.

The configuration for the i224atp(7iref) driver is handled by the same data structures as the i224a driver. The i224a driver must be configured into the system in order to configure the i224atp(7iref) driver.

Minor Table

The minor table maps a logical device's minor number to a partition on a physical device. A physical device consists of a specific unit on a board. The following table shows how the minor numbers are mapped onto the list of possible boards and units. For more information on minor number assignments, see i224aminor in the file /etc/conf/pack.d/i224a/space.c.

Board Table

The board table is an array of structures containing the information required to access each board in the system. A board is selected by the board index in the minor table. Each entry in this table contains the information required to communicate with the board, and a pointer to the drive table for each device connected to the board. See i224acfg in the file /usr/include/sys/i224a.h.

The drive table and partition table contain lists of the possible configurations for the driver. If you are configuring the driver, do not modify the contents of either the drive table or the partition table.

Hard Disk Information

This device driver requires certain information about a disk drive in order to access the drive. This information is stored on the disks themselves. hard disk device drivers(5iref) discusses the disk drive information that this and other disk device drivers need. Also, certain ioctl() calls are supported by all MULTIBUS disk device drivers. These calls are listed in the hard disk device drivers(5iref) man page.

Minor Table	Unit	Board
0 - 15	Wini0	0
16 - 31	Wini1	0
32 - 47	Wini2	0
48 - 63	Wini3	0
64 - 78	Floppy0	0
79 - 93	Floppy1	0
94 - 108	Floppy2	0
109 - 123	Floppy3	0
124,125	Tape0	0
126,127	Tape1	0
128 - 143	Wini0	1
144 - 159	Wini1	1
160 - 175	Wini2	1
176 - 191	Wini3	1
192 - 206	Floppy0	1
207 - 221	Floppy1	1
222 - 236	Floppy2	1
237 - 251	Floppy3	1
252,253	Tape0	1
254,255	Tape1	1

Flexible Disk Information

There is a device file in the /dev directory for each type of valid flexible disk drive. The device filenames follow a naming convention that is discussed in the `flexible disk device drivers`(5iref) man page.

FILES

/usr/include/sys/i224a.h
/etc/conf/pack.d/i224a/space.c

SEE ALSO

`flexible disk device drivers`(5iref)
`hard disk device drivers`(5iref)
`i214`(7iref), `i224atp`(7iref), `i258`(7iref)
iSBCR 186/224A Peripheral Controller Board User's Guide

NAME

`i224atp` – iSBC 186/224A peripheral controller tape device driver

DESCRIPTION

The `i224atp` device driver provides an interface to tape devices controlled by the PCI running on an iSBC 186/224A peripheral controller board. The driver supports 1/4-inch QIC-02 cartridge magnetic tape only.

The configuration for the `i224atp` driver is handled by the same data structures as the `i224a` driver. The `i224a` driver must be configured into the system in order to configure the `i224atp` driver.

Tape device filenames follow a naming convention that is discussed in the `tape device drivers`(5iref) man page.

FILES

`/etc/cinf/pack.d/i224a/space.c`
`/usr/include/sys/i224a.h`

SEE ALSO

`tape device drivers`(5iref), `i224a`(7iref)

NAME

i258 – iSBC 386/258 peripheral controller device driver

DESCRIPTION

The i258 device driver provides an interface to hard disk and flexible disk devices controlled by the Peripheral Controller Interface (PCI) running on an Intel iSBC 386/258 peripheral controller board. (For cartridge tape support, see i258tp(7iref).) The iSBC 386/258 board supports the SCSI device interface.

This driver supports up to four hard disk drives and two diskette drives on one controller board. The driver supports one or two controller boards, with two boards as the default configuration.

The interface to the i258 driver is the standard System V/386 block and character device interface.

CONFIGURATION

The i258 driver communicates with the PCI, which in turn communicates with SCSI devices. The PCI and SCSI interfaces address devices differently. When configuring the driver, use the PCI unit addressing scheme to address devices; the PCI translates addresses received from the driver to the proper SCSI addresses. For information on this address translation, see the *How To Use The Peripheral Controller Interface (PCI) Server.*

Configuration for the i258 driver is handled by the following data structures: the minor table, the board table, the drive table, and the partition table.

Minor Table

The minor table maps a logical device's minor number to a partition on a physical device. A physical device consists of a specific unit on a board. The unit numbers in the i258 minor table are the PCI unit numbers for the specified devices. The table on the next page shows how the minor numbers are mapped onto the list of possible boards, devices, and PCI units. For more information on minor number assignments, see i258minor in the file /etc/conf/pack.d/i258/space.c.

Board Table

The board table is an array of structures containing the information required to access each board in the system. A board is selected by the board index in the minor table. Each entry in this table contains the information required to communicate with the board, and a pointer to the drive table for each device connected to the board. See i258cfg in the file /usr/include/sys/i258.h.

The drive table and partition table contain lists of the default configurations for devices controlled by the driver. When configuring the i258 driver, you may want to add devices with different characteristics than the default devices.

The following steps provide an overview of the procedure you must follow to add a new device:

Add an entry into the drive table for the new device type. The drive table is in the file /etc/conf/pack.d/i258/space.c. Hard disks drives are defined in the structure i258cdrt i258d00[], diskette drives in the structure i258cdrt i258f00[], and tape drives in the structure i258cdrt i258t00[]. Add an entry to the partition table for each partition on the new device type. The `partition table` is in the file `/etc/conf/pack.d/i258/space.c`. Partitions for diskette drives are defined in the structure `i258cdrt i258Pf0[]`. Partitions for tape drives are not configurable. Partition information for hard disks is determined at runtime from the volume table of contents (vtoc) on the drive. (See `/usr/include/sys/vtoc.h` for information on the vtoc.) Put an entry into the `minor table` for each partition of the new device. You may use the minor numbers listed as unsupported in the previous table, or replace a device currently in the minor table if you will no longer be using that device. Use the mknod(1M) command to create a node in the `/dev` directory. Follow the naming conventions for the various device types listed in the `hard disk drive devices`(5iref), `flexible disk drive devices`(5iref), and `tape drive devices`(5iref) man pages. Be sure that the minor number you use in the mknod(1M) command is the same as the minor number in the minor table for the desired partition.

Minor Table	Device	PCI Unit	Board
0 - 15	Wini0	2	0
16 - 31	Wini1	3	0
32 - 47	Wini2	4	0
48 - 63	Wini3	5	0
64 - 78	Floppy0	0	0
79 - 93	Floppy1	1	0
94 - 108	unsupported		0
109 - 123	unsupported		0
124,125	Tape0	6	0
126,127	unsupported		0
128 - 143	Wini0	1	
144 - 159	Wini1	1	
160 - 175	Wini2	1	
176 - 191	Wini3	1	
192 - 206	Floppy0	1	
207 - 221	Floppy1	1	
222 - 236	unsupported		1
237 - 251	unsupported		1
252,253	Tape0	1	
254,255	unsupported		1

An example later in this driver description details the above procedure.
The partition table is in the `i258part` structure in the file `/etc/conf/pack.d/i258/space.c`.
The configuration for the `i258tp`(7iref) driver is handled by the same data structures as the `i258` driver. The `i258` driver must be configured into the system in order to configure the `i258tp`(7iref) driver.

This device driver requires certain information about a disk drive in order to access the drive. This information is stored on the disks themselves. **hard disk device drivers**(5iref) discusses the disk drive information that this and other disk device drivers need. Also, certain `ioctl()` calls are supported by all MULTIBUS disk device drivers. These calls are listed in the **hard disk device drivers**(5iref) man page.
There is a device file in the **/dev** directory for each type of valid flexible disk drive. The device filenames follow a naming convention that is discussed in the **flexible disk devicedrivers**(5iref) man page.

EXAMPLE

Here is an example of adding a 3.5-inch diskette drive as unit 0 on board 0 to the i258 driver configuration.

1. Add the following to the **i258cdrt i258f00[]** structure in the **/etc/conf/pack.d/i258/space.c** file:

```
/* 3.5 inch -1.44 Mbyte capacity  FD3_7958BR_135TPI */
80, 2, 18,  512, I258FLP(3, 0, 0, 9), 0, i258Pf0 /* [8] */
```

2. Add the following to the **i258cdrt i258Pf0[]** structure in the **/etc/conf/pack.d/i258/space.c** file:

```
/* 3.5 inch 1.44 Mbyte capacity  Hds-2 Cyls-80 Secs-18 Bytes-512 */
0,V_VOMASK,  0, 2880, /* [40] Whole disk */
0,V_VOMASK, 36, 2844, /* [41] Skip track/cylander 0 */
0,       0,  0,    0, /* [42] Future use */
0,       0,  0,    0, /* [43] Future use */
0,       0,  0,    0, /* [44] Future use */
```

3. Add the following to the **i258minor** structure in the **/etc/conf/pack.d/i258/space.c** file:

```
/* Board 0 - Floppy 0 */
i258MINOR(0,0,8,40)          /* [ 94] 1.44 Mbytes - f03ht */
i258MINOR(0,0,8,41)          /* [ 95] 1.44 Mbytes - f03hu */
```

4. Use the following commands to create the device nodes for the new diskette drive in the proper device directories:

```
/etc/mknod  /dev/dsk/f03ht   b  0  94
/etc/mknod  /dev/dsk/f03hu   b  0  95
/etc/mknod  /dev/rdsk/f03ht  c  0  94
/etc/mknod  /dev/rdsk/f03hu  c  0  95
```

FILES

/usr/include/sys/i258.h
/etc/conf/pack.d/i258/space.c

SEE ALSO

`flexible disk device drivers`(5iref) `hard disk device drivers`(5iref) `i224a`(7iref), `i214`(7iref), `i258tp`(7iref)

NOTES

The `i258` driver only supports two diskette drives and one tape drive. The `i258` driver only supports one tape drive.

NAME

i258tp – iSBC 386/258 peripheral controller tape device driver

DESCRIPTION

The i258tp device driver provides an interface to tape devices controlled by the PCI running on an iSBC 386/258 peripheral controller board. The iSBC 386/258 board supports the SCSI device interface for tapes.

The configuration for the i258tp driver is handled by the same data structures as the i258 driver. The i258 driver must be configured into the system in order to configure the i258tp driver.

Tape characteristics are defined in the i258cdrt i258t00[] structure in the file /etc/conf/pack.d/i258/space.c. The device filenames follow a naming convention that is discussed in the tape device drivers(5iref) man page. The device nodes created in /dev/rmt must match the entries in the minor table in the file /usr/include/sys/i258.h.

SEE ALSO

tape device drivers(5iref), i258(7iref)

NAME

`i354` – iSBX 354 dual channel serial-port device driver

DESCRIPTION

The `i354` device driver, together with the `iasy`(7iref) driver, provides a termio interface to the iSBX 354 dual channel serial controller board on MULTIBUS II systems. The board contains the Intel 82530 Serial Communications Controller providing two serial channels to the local processor on which the board is mounted.

The `i354` device driver contains support for the console device (as per `console`(7iref)). The `i354` will automatically be configured into the data structure **conssw** (this data structure is used by `console`(7iref)).

FILES

`/etc/conf/cf.d/mdevice`	mdevice entries
`/etc/conf/sdevice.d/i354`	sdevice entries
`/etc/conf/pack.d/i354/Driver.o`	`i354` device driver object module
`/usr/include/sys/i354.h`	`i354` specific definitions
`/usr/include/sys/i82530.h`	`i82530` SCC specific definitions
`/etc/conf/pack.d/i354/space.c`	`i354` specific declarations and

SEE ALSO

`console`(7iref), `iasy`(7iref), `termio`(7)

NAME

`i410` – iSBC 186/410 serial communications controller device driver

DESCRIPTION

The `i410` device driver performs recognition and initialization of iSBC 186/410 serial communications controller boards at system boot-up.

At boot time, the number of iSBC 186/410 serial communications controller boards present in the MULTIBUS II system is determined. If any are present, they are examined to determine if they are booting properly. Information describing the state of any iSBC 186/410 serial communications controller boards in the system is displayed on the console at boot-time.

FILES

`/etc/conf/cf.d/mdevice`	mdevice entries
`/etc/conf/sdevice.d/i410`	sdevice entries
`/etc/conf/pack.d/i410/Driver.o`	`i410` device driver object module
`/usr/include/sys/ics.h`	interconnect space definitions
`/etc/conf/pack.d/i410/space.c`	`410` specific declarations and initialization

SEE ALSO

`atcs`(7iref), `cci`(7iref)

NAME

`i530` – iSBC 186/530 Ethernet controller device driver

DESCRIPTION

The `i530` driver provides a STREAMS interface to the iSBC 186/530 Ethernet controller board. Applications access the driver through the UNIX Transport Interface (also know as the Transport Layer Interface or TLI). The TLI NSL library enables users to bind an address to an endpoint, listen for connect requests, accept connect requests, send and receive data, and so on.

TLI applications normally call the **t_open()** function as the first step in initialization of transport endpoints. Alternatively, the **n_connect()** or **n_accept()** SV-OpenNET NSI library calls may be used to perform this initialization. A successful **t_open** call returns a file descriptor that references a STREAMS path to the transport provider, and typically, an allocated endpoint structure. The first argument to **t_open** is a path name that identifies the agent that will provide the transport service.

As a transport provider, the `i530` driver maintains a table of endpoint structures indexed by minor number. Elements of this table may be allocated and used exclusively or may be shared by TLI applications. Applications target the `i530` driver in one of two ways. First, the application may supply the device node **/dev/iso-tp4** as the **t_open** path name. This action routes the call through the clone driver. The clone driver finds an available `i530` minor number and the associated virtual circuit endpoint structure for the application to use. Alternatively, the application may access the driver directly by supplying the **t_open** pathname **/dev/tp4-***xx* where *xx* represents a digit from the set 01 through the maximum configured endpoints plus one (31 by default). This way, applications can target specific endpoints and share them with other applications, if desired. Though available as a transport endpoint, the path name assigned to minor device 0 (**/dev/tp4-00**) is typically reserved for driver administration and control, and should NOT be used by normal TLI applications.

Some TLI commands, such as **t_bind()** and **t_connect()**, target a local or remote endpoint via a "driver-ready" transport address. The address on the listener side of a connection must be externally known and typically is obtained by the TLI application from a name server. This name server translates a name of a transport service endpoint into a binary format usable by the driver. The transport address used by the `i530` driver (via the iNA 960 Network Layer) is a 15 byte address containing the NSAP (network service access point) which includes the ethernet address of the ethernet controller and the TSAP (transport service access point) selector for the endpoint in the following form:

0B 49 00 01 *XX XX XX XX XX XX* FE 00 02 *YY YY*

where *XX ... XX* represents the ethernet address of the controller (e.g. 00AA00029CB9) and *YY YY* represents the TSAP selector (a two byte number that defines the point of access to a client process) for the endpoint. The ethernet address for the iSBC 186/530 controller in a system is obtained by executing the enetinfo utility.

Configuration and tunable parameters for the i530 driver are defined in /etc/conf/pack.d/i530/space.c. In order to change the driver configuration, this file must be edited and the kernel regenerated. These parameters are described below:

N_ENET	Number of 186/530 boards in the system. The default is 1.
NVC	Maximum number of exclusive device opens. The default is 31.
N_ENDPOINTS	Maximum number of endpoints that the i530 driver supports. The default is 31. The maximum is 101.
DATA_BUF_LEN	Buffer size for receiving incoming messages. The default is 4096 bytes.
MAX_BUFS_	Maximum number of buffers posted for POSTED receiving data. The default is 1.
MAXCOPYSIZ	Message size below which the data is copied into a new streams buffer. For messages below this size, a new streams buffer of that size is allocated. This is for beeter utilization of the various streams sizes. The default is 128 bytes.
MAX_DATA_RQ	Maximum number of send data requests that can be serviced by the stream. The default is 1.
SH_HIWAT	When the total number of characters in all the messages that are queued exceeds SH_HIWAT, messages from upstream are halted.
SH_LOWAT	When the total number of characters in all the messages that are queued is below SH_LOWAT, messages from upstream are enabled.

EXAMPLE

The following commands can be used to setup the i530 driver as a network listener for RFS.

```
nlsadmin -i iso-tp4
nlsadmin -a 105 -c /usr/net/servers/rfs/rfsetup -y "RFS server" iso-tp4
nlsadmin -l "49000100AA00029CB9FE000200A0" iso-tp4
nlsadmin -s iso-tp4
```

This example assumes that the ethernet address for the iSBC 186/530 Ethernet controller is 00AA00029CB9.

FILES

/etc/conf/pack.d/i530/space.c	i530 driver configuration file
/etc/conf/pack.d/i530/Driver.o	i530 driver object file
/etc/conf/node.d/i530	i530 driver device node definition
/etc/conf/sdevice.d/i530	i530 driver system device entry
/usr/include/sys/enetuser.h	i530 driver user data structures

SEE ALSO

enetinfo(1iref)

NAME

`ics` – Interconnect Space device driver

DESCRIPTION

The `ics` driver handles the interface between MULTIBUS II device drivers or applications and MULTIBUS II Interconnect Space as implemented by the MPC component.

To configure the `ics` driver, modify the following variables in the `/etc/conf/pack.d/ics/space.c` file:

ics_hi_addr The I/O port address for selecting the high-order eight bits of the interconnect address.

ics_low_addr
The I/O port address for selecting the low-order eight bits of the interconnect address.

ics_data_addr
The I/O port address for data to be read or written to the interconnect register.

ics_cpu_cfglist
A list of all board types on which UNIX may be running in the system. This list must be terminated with a NULL pointer.

ics_bdev A table listing root, swap, pipe, and dump devices for each possible UNIX processor in the system.

ics_max_numcpu
The maximum number of UNIX processor boards permitted in a system. This must correspond to the number of entries in the `ics_bdev` table.

Following is a list of the supported `ioctl()` calls provided by the `ics` driver:

`Read Interconnect Space`(ICS_READ_ICS)
This command reads Interconnect Space. It takes a pointer to a `struct ics_rdwr_args` and does the Interconnect Space read operation encoded therein.

`Write Interconnect Space`(ICS_WRITE_ICS)
This command writes Interconnect Space. It takes a pointer to a `struct ics_rdwr_args` and does the Interconnect Space write operation encoded therein.

`Find Interconnect Space Record`(ICS_FIND_REC)
This command finds a specific Interconnect Space record. It takes a pointer to a `struct ics_frec_args` and does the Interconnect Space find record operation encoded therein.

`Get Current Slot ID`(ICS_MY_SLOTID)
This command returns the slot ID of the current slot. It takes a pointer to an `unsigned char` which is set to the Interconnect Space slot id of the caller.

`Get Current CPU Number`(ICS_MY_CPUNUM)
This command returns the CPU number of the current UNIX processor. It takes a pointer to an `unsigned char` which is set to the Interconnect Space CPU number of the caller. The CPU number is an identifier for this particular processor.

`Get Current CPU Number`(ICS_MY_CPUNUM)
: This command returns the CPU number of the current UNIX processor. It takes a pointer to an `unsigned char` which is set to the Interconnect Space CPU number of the caller. The CPU number is an identifier for this particular processor.

The following is a list of error conditions that may be returned:

`EFAULT`	A memory address used in an ioctl was not a valid data address.
`EINVAL`	An Interconnect space address used in an `ioctl` was not valid, or an invalid `ioctl` request was attempted.

FILES

`/usr/include/sys/ics.h`	definitions for data structures
`/etc/conf/pack.d/ics/space.c`	configuration information

SEE ALSO

MULTIBUSR II Interconnect Interface Specification
System V/386 Device Driver Interface/Driver-Kernel Interface Reference Manual

NOTES

Specifying *ics_hi_addr:ics_low_addr* gives the address of the interconnect register.

NAME

mpc – MPC device driver

DESCRIPTION

The mpc driver handles the internal low level interface between the ics(7iref) and mps(7iref) interface and the actual message passing coprocessor (MPC) hardware of a MULTIBUS II CPU board.

To configure the mpc driver, modify the following variables in the `/etc/conf/pack.d/mpc/space.c` file:

impc_base Contains the base I/O port address for the MPC chip.

impc_fs_enabled
Contains a flag indicating whether or not the MPC fail-safe mechanism should be used. It is strongly recommended that the fail-safe mechanism be enabled.

There are no `ioctls` for the mpc driver. The mpc driver is used to support the internal interface required by the mps(7iref) driver.

FILES

`/etc/conf/pack.d/mpc/space.c` configuration information

SEE ALSO

ics(7iref), mps(7iref) mps(7iref)

NAME

mps – Message Passing Space device driver

DESCRIPTION

The **mps** driver handles the interface between MULTIBUS II device drivers or applications and MULTIBUS II message passing space as implemented by the MPC component.

To configure the **mps** driver, modify the following variables in the `/etc/conf/pack.d/mps/space.c` file:

mps_max_tran
: The maximum number of outstanding transactions to be allowed. This number must correspond to the number of entries in the `mps_tinfo` and `mps_t_ids` arrays declared in the `space.c` file.

mps_max_port
: The maximum number of open ports to be allowed. This number must correspond to the number of entries in the `mps_port_defs` and `mps_port_ids` arrays declared in the `space.c` file.

The **mps** driver does not support any `ioctl` calls for use by applications. Instead, it supports the Message Passing interface required by the System V/386 Device Driver Interface specification, which is used by other device drivers.

FILES

`/etc/conf/pack.d/mps/space.c` configuration information

SEE ALSO

MULTIBUS II Message Passing Coprocessor External Product Specification MPS User's Manual
System V/386 Device Driver Interface/Driver-Kernel Interface Reference Manual
ots(7iref)

NAME

ots – System V/386 OSI Transport Service (ots) device driver

DESCRIPTION

The ots driver is a STREAMS device driver that allows applications executing on separate MULTIBUS II processors to establish connections and exchange data messages with one another. ots can also be used between applications on the same processor. Applications access the driver through the UNIX Transport Interface (also known as the Transport Layer Interface or TLI). Both virtual circuit (VC) and datagram services are provided by ots. The ots driver only supports message exchange between processors in the same MULTIBUS II system.

TLI applications normally call the t_open() function as the first step in initialization of transport endpoints. Alternatively, the n_connect() or n_accept() SV-OpenNET NSI library calls may be used to perform this initialization. A successful t_open call returns a file descriptor that references a STREAMS path to the transport provider and typically, an allocated endpoint structure. The first argument to t_open is a path name that identifies the agent that will provide the transport service.

As a transport provider, the ots driver maintains a table of endpoint structures indexed by minor number. Elements of this table may be allocated and used exclusively or may be shared by TLI applications. Applications target the ots driver in one of two ways. First, the application may supply the device nodes /dev/ots-vc or /dev/ots-dg as the t_open path name. This action routes the call through the clone device driver. The clone driver finds an available ots minor number and associated endpoint structure for the application to use. When /dev/ots-vc is selected, the driver allocates a virtual circuit endpoint. Access to the driver through /dev/ots-dg causes the allocation of a datagram endpoint. Alternatively, the application may access the driver directly by supplying the t_open path name /dev/ots-*xx* where *xx* represents a digit from the set 01 through the maximum number of configured endpoints plus one (31 by default). This way, applications can target specific endpoints and share them with other applications, if desired. Though available as a transport endpoint, the path name assigned to minor device 0 (/dev/ots-00) is typically reserved for driver administration and control and should NOT be used by normal TLI applications (This is in contrast to the path name used by the Ethernet driver-- /dev/iso-tp4).

Some TLI commands, such as t_bind() and t_connect(), target a local or remote endpoint via a *driver-ready: transport address. The address on the listener side of a connection must be externally known and typically is obtained by the TLI application from a name server. This name server translates a name of a transport service endpoint into a binary format usable by the driver. The transport address expected by ots is a MULTIBUS II socket. The binary format of the socket is a double word consisting of the host ID in the high-order word and port ID in the low-order word. The host ID identifies the processor on which the transport endpoint resides. The port ID identifies the transport service executing on that processor. Port ID's utilized by the ots driver range in value between 0x300 and 0x4FF. The following table defines those ports within this range that target well-known services, are available for use by custom TLI applications and are reserved for exclusive use by ots.

Port ID	Description
0x300	System V/386 general listener service
0x301	System V/386 terminal login service
0x302 to 0x3FF	User-defined well-know ports
0x400	SV-OpenNET NFA Server
0x401	SV-OpenNET NFA Consumer
0x402	SV-OpenNET Virtual Terminal Server
0x403	SV-OpenNET Virtual Terminal Consumer
0x404 to 0x41F	Reserved for future SV-OpenNET services
0x420 to 0x4FF	Reserved by SV-ots driver

By convention, the host ID is typically set to the processor's slot number within the MULTIBUS II backplane. The `ots` driver uses the host ID to distinguish local from remote addresses. If the host ID portion of a socket is the same as the processor's slot ID, the referenced address is assumed local. When the driver receives a connection request that targets a local listening endpoint, it routes the request back up the listening stream. This action generates no bus activity. Thus, the `ots` driver may be used for message exchange between TLI applications executing on the same processor.

The `ots` driver recognizes the following protocol options which the transport user may retrieve, verify, or modify via t_optmgmt():

`OPT_COTS` Connection-Oriented Transport Service (i.e. virtual circuit service). This option can NOT be modified by a TLI application. The service is automatically assigned when the device /dev/ots-vc is opened. VC service on an endpoint is indicated if bit 1 is set in the option field.

`OPT_EXP` Expedited message delivery service. This is selected by setting bit 2 in the option field.

`OPT_ORD` Orderly disconnect or release on a virtual circuit. This is selected by setting bit 3 in the option field.

`OPT_CLTS` Connectionless Transport Service (i.e. datagram service). This option cannot be modified by a TLI application. The service is automatically assigned when the device `/dev/ots-dg` is opened. Datagram service on an endpoint is indicated if bit 4 is set in the option field.

The option field is a double word and can contain any logical combination of the above bit fields. These macros are defined in the header file `/usr/include/sys/otsuser.h`.

Configuration and tunable parameters for the `ots` driver are defined in `/etc/conf/pack.d/ots/space.c`. In order to change the driver configuration, this file must be edited and the kernel regenerated. These parameters are described below:

`NVC` Maximum number of Virtual Circuits endpoints supported by the `ots` driver. The default is 25. With this default, virtual circuit endpoints can be accessed directly via the device nodes `/dev/ots-01` through `/dev/ots-25`.

NDG
: Maximum number of Datagram endpoints that the ots driver supports. The default is 5. Assuming default values for NVC and NDG, datagram endpoints can be accessed directly via the device nodes /dev/ots-26 and /dev/ots-30.

MAX_PEND
: Maximum number of concurrent connection indications the ots driver will accept on a listening endpoint. The default is 5.

TSDU_SIZE
: A value greater than zero specifies the maximum size of a transport service data unit (TSDU or a message); a value of zero specifies that the ots driver will not support the concept of a TSDU. The default value is 64 Kbytes. The largest value which may be configured is 16 Megabytes.

ETSDU_SIZE
: A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of an ETSDU. The default value is 0 bytes. The largest value which may be configured is 4096 bytes.

CDATA_SIZE
: A value greater than zero specifies the maximum amount of data that may be associated with the connection establishment functions. The default is 64 bytes. The largest value which may be configured is 512 bytes.

DDATA_SIZE
: Maximum amount of data that may be associated with disconnect requests. The default is 64 bytes. The largest value which may be configured is 512 bytes.

DATAGRAM_SIZE
: The maximum size of a datagram that may be sent by the transport user. The default and the largest value which may be configured is 4096 bytes.

OTI_RD_HIWAT
: When the total number of characters in all the messages queued at the driver's upstream queue exceeds OTI_RD_HIWAT (1024 bytes by default), the driver will stop receiving messages from remote endpoints.

OTI_RD_LOWAT
: When the total number of characters in all the messages queued falls below OTI_RD_LOWAT (512 bytes by default), the driver will again start receiving messages.

OTI_WR_HIWAT
: When the total number of characters in all the messages queued at the driver's downstream queue exceeds OTI_RD_HIWAT (1024 bytes by default) STREAMS blocks messages directed downstream.

OTI_WR_LOWAT
: When the total number of characters in all the messages queued falls below OTI_RD_LOWAT (512 bytes by default), STREAMS enables any downstream queue that had been blocked.

U_VCDEFAULTS
: This subset of driver options is applied to every newly initialized VC endpoint. Options are defined by bit fields in the file /usr/include/sys/otsuser.h.

U_DGDEFAULTS
: This subset of driver options is applied to every newly initialized datagram endpoint. Options are defined by bit fields in the file /usr/include/sys/otsuser.h.

If the total number of configured endpoints (NVC + NDG) is increased, device nodes for the additional endpoints should be created. This is done automatically by adding entries to the /etc/conf/node.d/ots.

Under heavy loads, the ots driver is potentially an intensive user of System V/386 Transport-Kernel Interface resources. Depending on the use of the ots driver, more space may need to be allocated for specific Transport-Kernel Interface tables and other data structures. Table sizes for the Transport-Kernel Interface module are defined as /etc/conf/pack.d/mpc/space.c in the configuration file. The following parameters should be increased two times (2X) the number of endpoints configured in /etc/conf/pack.d/ots/space.c.

Parameter	Description
MAX_PORT	Maximum number of open ports
MAX_TRAN	Maximum number of outstanding transactions

EXAMPLE

The following commands can be used to setup ots as a network listener for RFS.

```
nlsadmin -i ots-vc
nlsadmin -a 105 -c /usr/net/servers/rfs/rfsetup -y "RFS server"
nlsadmin -l "\x00030500" ots-vc
```

This example assumes the MULTIBUS II processor on which the listener is being setup is located in slot 5 of the MULTIBUS II backplane. Thus, the host ID is 0x0005. Because the port id assigned, the general listener service is 0x0300 and the socket address for this particular service is the double-word, 0x00050300. However, because the address is stored by the name server listener service (nls) as a character string, the socket's words, and the bytes within these words must be swapped.

FILES

/etc/conf/pack.d/ots/space.c	ots driver configuration file
/etc/conf/pack.d/ots/Driver.o	ots driver object file
/etc/conf/node.d/ots	ots driver device node definition file
/etc/conf/sdevice.d/ots	ots driver system device entry
/usr/include/sys/otsuser.h	ots driver user data structures

NAME

`rci` – debug console/rci protocol driver

DESCRIPTION

The `rci` driver, together with the `iasy`(7iref) driver, provides a debug console interface to allow kernel I/O on a device running an `rci` server. An example of a suitable `rci` server runs on the iSBC 258 disk controller when an iSBX 279 is attached.

If an `rci` server is running the kernel debug console, a suitable ATCS server must be running on the same host. The ATCS server provides a termio interface for `/dev/console` via the `iasy`(7iref) and `atcs`(7iref) drivers. A line from the ATCS server, corresponding to the current CPU number, will automatically be configured into the `conssw` data structure used by `console`(7iref).

FILES

`/etc/conf/cf.d/mdevice`	System configuration file
`/etc/conf/sdevice.d/rci`	`rci` configuration file
`/usr/include/sys/rcimp.h`	Protocol specific definitions

SEE ALSO

`termio`(7), `atcs`(7iref), `console`(7iref), `iasy`(7iref)

7. SYSTEM MAINTENANCE COMMANDS

7. SYSTEM MAINTENANCE COMMANDS

7 System Maintenance Commands

Maintenance Commands

This chapter lists Intel's System V/386 system maintenance commands. This chapter applies only to MULTIBUS II.

- dbon
- icsrd, icswr, icsslot, icsgetrec
- initbp
- reset

NAME

dbon – sets target for front panel message delivery

SYNOPSIS

`/usr/lbin/dbon` [*slot_ID*]

DESCRIPTION

The **dbon** command instructs the Central Services Module (CSM) where future Front Panel Interrupt Messages are to be sent. This is used for determining which processor will break into the kernel debugger when the Front Panel Interrupt is issued.

Setting the CSM to interrupt one's own processor is as follows. In order to instruct the CSM to interrupt the processor you are using, execute **dbon** with no arguments. The current slot number is determined automatically, and given to the CSM as destination of future front panel interrupt messages.

Setting the CSM to interrupt another processor is as follows. In order to instruct the CSM to interrupt another processor, **dbon** is executed with the slot number of the desired processor as the argument. Valid slot numbers are 0 through 21.

FILES

```
/usr/lbin/dbon
/sbin/icswr
/sbin/icsrd
/sbin/icsslot
```

SEE ALSO

`icswr`(8iref), `icsrd`(8iref), `icsslot`(8iref) `icsrd`(8iref), `icswr`(8iref), `icsslot`(8iref)

NOTES

This program is only effective on sytems with a CSM/001.

NAME

`icsrd`, `icswr`, `icsslot`, `icsgetrec` – utilities to access MULTIBUS II interconnect space

SYNOPSIS

`/sbin/icsrd` [*-h -s -d*] *slot_ID register count*
`/sbin/icswr` *slot_ID register count value*
`/sbin/icsslot`
`/sbin/icsgetrec` [*-h*] *slot_number record_type*

DESCRIPTION

The `ics` utilities provide a command level interface to MULTIBUS II Interconnect Space. These commands are not intended to provide a user interface; they are simply a base set of access functions upon which shell scripts may be easily written.

The command `icsrd` is used to read Interconnect Registers. The arguments to `icsrd` are a slot number, a starting register number, and a count of registers to be read. All arguments are assumed to be decimal (valid slot numbers are 0 through 21). Output consists of one line for each register, displayed on stdout. Each line consists of the following components:

```
00:002 (002H) -  67 (43H) [C]
```

The first two fields, separated by a colon, are the slot and register numbers. The third field is the register number, reprinted in hexadecimal notation. The fourth, fifth, and sixth fields are all representations of the value in the register. In order, they are: decimal, hex, and ASCII. The ASCII field is only printed if it is a printable ASCII character.

The options *-h*, *-s* and *-d* override the default output shown above. These options are mutually exclusive. The *-h* option outputs hex values. The *-d* option outputs decimal values. The *-s* option outputs the values as a string. For example,

```
icsrd -s 1 2 10
```

will print out the board-id of the board in slot 1 (such as the iSBC 186/410).

The command `icswr` is used to write Interconnect Space Registers. The arguments to `icswr` are a slot number, a starting register number, a count, and a value. All arguments are assumed to be in decimal. There is no output.

The command `icsslot` returns the slot number of the processor it was executed on. By using `icsslot`, the user can determine which processor they are using. This command also provides shell scripts a way of finding out which slot they are being run on.

Finding a specific record in the interconnect space of a board:

The command `icsgetrec` returns the starting register number of the specified record type. The arguments to `icsgetrec` are a slot number and a record type. If such a record is found, its starting register number is returned. If the interconnect register cannot be found in the specified slot, a −2 is returned. For all other errors, a −1 is returned. All arguments are assumed to be decimal (valid slot numbers are 0 through 21 and valid record types are 0 through 255). Output is in decimal format, unless the −h option is used to specify a hexadecimal value.

For example, if you wanted the starting register number of the Firmware Communications Record (record type 15) of a serial board in slot 6, you would use the following command:

```
icsgetrec 6 15
```

FILES

```
/sbin/icswr
/sbin/icsrd
/sbin/icsslot
/sbin/icsgetrec
```

SEE ALSO

dbon(8iref), `reset`(8iref)

NOTES

The user interface of these commands is not easy to use. These utilities are intended as building blocks for shell scripts to manage interconnect space registers. Also, care should be taken when using `icswr`; with it any board on the bus can be reset, resulting in loss of data.

NAME

`initbp` – initializes the bootstrap parameter string on the processor in the given slot

SYNOPSIS

`/usr/lbin/initbp` [−*v*] *slot-ID*

DESCRIPTION

The `initbp` command initializes the bootstrap parameter string on a processor to a NULL string. This places the processor into a state similar to that which exists after a cold reset. All bootstrap parameters from all sources (the configuration file or those supplied from the operator interface) are deleted.

The `initbp` command resets the processor in the specified slot after the initialization of the bootstrap parameter string is complete. Command line options are:

−*v* Display diagnostic information.

FILES

`/usr/lbin/initbp`
`/usr/lbin/reset`

SEE ALSO

`ics`(8iref), `dbon`(8iref), `reset`*(8iref)*

NOTES

For additional information on the initialization of MULTIBUS II processor boards, see the *MULTIBUSR II Initialization and Diagnostics Specification*. Also note that any processor can be reset when using `initbp`, resulting in a loss of data.

NAME

`reset` – resets the processor in a given slot

SYNOPSIS

`/usr/lbin/reset` [`-b`] [`-v`] [`-m` `-n` `-i` index] *slot-ID*

DESCRIPTION

The **`reset`** command issues a local reset to the processor in a given slot of the bus through interconnect space. This allows the operator to shut down one of the processors in a system and reboot it, without being required to shut the entire system down. Valid slot numbers are 0 through 20. Command line options are:

`-b`	The BIST complete bit in the BIST master status register is not set. Default is set to the bit.
`-v`	Verbose. Diagnostic information is displayed during execution of the reset command.
`-m`	The contents of the program table index register (PTIR) are updated to invoke the debug monitor. This is the same as specifying `-i 248`.
`-n`	The contents of the PTIR are left unchanged. This allows the user to sequence through the entries in the program table.
`-i`	Set the value of the program table index register (PTIR) to index. The default is 0.

FILES

`/usr/lbin/reset`

SEE ALSO

`ics`(8iref), `dbon`(8iref), `initbp`(8iref)

NOTES

For additional information on the values of the program table index register (PTIR), see the *MULTIBUSR II Initialization and Diagnostics Specification.* Also note that when using the **`reset`** command, any processor can be reset; resulting in a loss of data.

8. STATIC BAD BLOCK HANDLING

8. STATIC BAD BLOCK HANDLING

8 Static Bad Block Handling

Overview

This chapter describes the Bad Block Handler software module for Intel's System V/386 Operating System. This chapter also includes information needed in creating drivers for MULTIBUS based systems.

The Bad Block Handler (BBH) is a set of routines that compensates for the unusable sectors (blocks) of a hard disk drive. Each bad block of a disk is remapped so that all references to the bad block are transparently diverted to a known usable block. This transparent redirection causes the disk to appear to be flawless even though it may contain several unusable blocks.

Architecture

Overview

The BBH compensates for the unusable blocks of a hard disk by transparently substituting a usable block for each bad block. This transparent redirection is actually performed by the controller hardware not the BBH software. Instead, the BBH manages the remapping process by telling the controller which usable block is to be substituted for each bad block on the disk.

Static bad blocks are remapped during the format phase of disk initialization. A list of static bad blocks is generated either from the Manufacturer's Defect List (see Appendix C) or from the user. As the disk is being formatted, each track containing a bad block is formatted using one of the remapping methods supported by the controller hardware. All future references to the bad block are redirected to the substitute block. BBH related structures and definitions are located in the file `/usr/include/sys/bbh.h`.

Disk Partitions

Each physical disk drive in the system must provide support for the Bad Block Handler. The BBH requires a few reserved sections of the media. Each disk drive must provide the following reserved sections:

Manufacturer's Defect List

The Manufacturer's Defect List (MDL) is a reserved portion of the disk that contains a list of the known bad blocks on the disk. The list includes static bad blocks and any dynamic bad blocks that have been remapped since the disk was last initialized. In most cases, the defect list is written on the disk prior to initialization. If the Manufacturer's Defect List is not written on the disk, a defect list will be created during the initialization procedure (see Appendix C for additional information).

Surrogate Block Pool

In order to remap a bad block to a known usable block, a portion of the disk must be set aside to form a pool of surrogate blocks. The various remapping methods allocate surrogate blocks using different methods and therefore require different pools of surrogate blocks. A disk driver that supports more than one remapping method requires more than one surrogate block pool. Each pool of surrogate blocks is used exclusively by the disk driver and not by any other component of the BBH. Following are three remapping methods.

Software Remapping

This remapping method uses a pool of individual blocks that are allocated one at a time. A software table is used to map the bad blocks to their surrogate blocks. The table is scanned during each I/O request with a match resulting in the referenced block being substituted with the previously assigned surrogate. Since the use of Software Remapping invalidates some of the functional requirements of the BBH, it is not intended to be used as a primary remapping method. However, it should be supported by all disk drivers, as a last resort, so the disk remains usable.

Alternate Tracking

This remapping method uses a pool of surrogate (alternate) tracks. Each track containing one or more bad blocks is remapped to one of the alternate tracks. The entire track, not just the blocks that are bad, is remapped. The remapping is performed by the controller hardware and is transparent to the disk driver software.

Sector Slipping

This remapping method requires that a certain number of blocks on each track is reserved for surrogate blocks. When a bad block is remapped, one of the reserved blocks within the same track is used as the surrogate. The number of bad blocks on a single track can not be greater than the number of reserved blocks within that track.

A. WINDOW INTERFACE

A. WINDOW INTERFACE

A Appendix A

Window Interface

The iSBX 279 provides the user of the System 520 with:

- A windowed environment
- A mouse to manipulate the windows
- A PC style keyboard for entering data

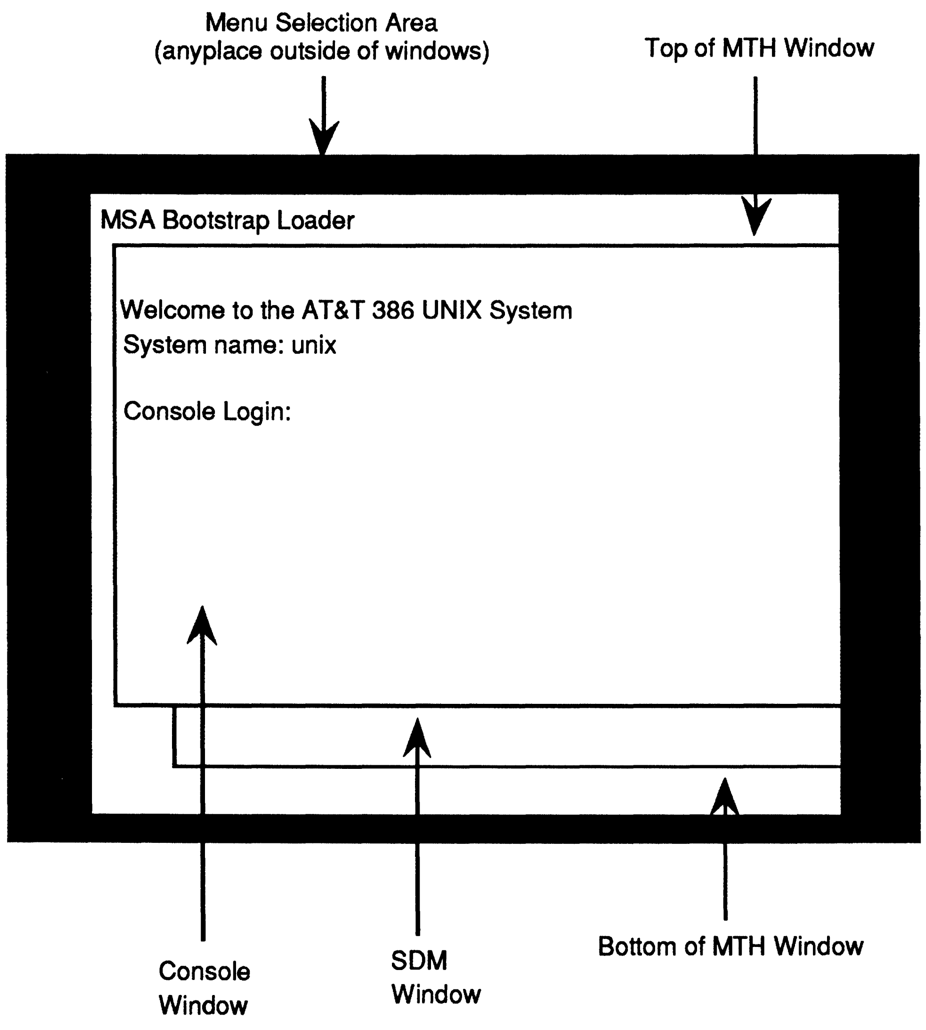
The following sections describe what the windows are and how to manipulate them with the mouse.

Windows, What are they?

The windows provided by the 279 are ways of viewing many operations simultaneously on one screen. They can be thought of as many terminals contained in one; each window representing a terminal. For example, the SYP 520 initializes with three windows: one for console input, one for displaying kernel messages and for using the debugger, and one for displaying messages from the Master Test Handler (MTH) (see Figure A-1). What would have required three terminals to view now requires one.

The windows can be manipulated with the mouse (moved, resized, and relayered) so that all or some of the windows can be viewed at once.

Figure A-1: An Example of Windows Displayed on the SYP 520



```
Welcome to the AT&T 386 UNIX System
System name: unix

Console Login:
```

Using The Mouse

The mouse can be used to move, resize, and relayer the windows. It can also be used to determine which window the keyboard will function with. All of these actions are provided in one of two pop-up menus, basic or expanded.

The basic menu appears before the System V/386 operating system is installed or initialized. Once installed, the operating system invokes the expanded menu during the boot up procedure. The expanded menu offers more options and provides a fast method of selecting windows (just point at the desired window and press any of the mouse buttons).

To select an option from either pop-up menu, do the following:

1. Move the mouse so that the pointer is outside any window.
2. Press and hold any one of the mouse buttons. The menu will appear on the screen.
3. Move the mouse up or down to select an option. Each option is highlighted as the pointer passes over it.
4. Release the mouse button when the desired option is selected.

NOTE

The mouse cannot be used to select anything inside of a window (a file for instance).

Basic Menu

This section explains how to use the selections provided by the basic menu.

The basic menu is shown in Figure A-2. This menu appears before the System V/386 operating system is installed or initialized. Once installed, the operating system invokes the expanded menu (explained later).

Figure A-2: Basic Menu Selections

Intel System Menu
Pop
Push
Pan
Move
Resize
Keyboard Focus
Pop/Focus
Pop/Focus/Resize
Exit

Pop:

This causes the selected window to appear on top of all other windows. The keyboard may not be "attached" to the window that has been popped. Use the `Keyboard Focus` selection to attach the keyboard to the popped window.

1. Select `Pop` from the menu with the mouse. The pointer will change to an up arrow.
2. Place the pointer within the desired window and press any of the mouse buttons.
3. The window selected will appear on top of all other windows.

Push:

This causes the selected window to be placed behind all other windows.

1. Select `Push` from the menu with the mouse. The pointer will change to a down arrow.
2. Place the pointer within the desired window and press any of the mouse buttons.

3. The window selected will be placed behind all other windows.

Pan:

This allows the contents of a window to be moved. This selection is useful for viewing the contents of a window that has been reduced in size.

1. Select `Pan` from the menu with the mouse. The pointer will appear as a square with four arrows pointing out from each side of the square.
2. Place the pointer within the desired window. Press and hold down any of the mouse buttons.
3. Moving the mouse will cause the contents of the window to move.
4. Release the mouse button when the desired contents are displayed.

Move:

This allows a window to be moved around the screen.

1. Select `Move` from the menu with the mouse. The pointer will appear as two crossed (up/down left/right) bidirectional arrows.
2. Position the pointer within the desired window. Press and hold down any of the mouse buttons.
3. Moving the mouse will cause the entire window to move.
4. Release the mouse button when the window is in the desired location.

Resize:

This selection causes the size of the window to be changed.

1. Select `Resize` from the menu with the mouse. The pointer will appear as two crossed bidirectional arrows, slightly tilted.
2. Position the pointer within the desired window. The pointer should be near one of the four corners of the window. The corner selected will be the part of the window that moves. Press and hold down any of the mouse buttons.

3. Move the mouse. This causes the size of the window to change.
4. Release the mouse button when the window is at the desired size.

NOTE Window contents are not rescaled when the window is resized. The maximum size of a window is either the size of the bitmap in which it is drawn or the size of the screen.

Keyboard Focus:

This selection allows keyboard input to be directed to a window.

1. Select `Keyboard Focus` from the menu with the mouse. The pointer will appear as a shaded box.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. Keyboard input is now directed to the selected window.

Pop/Focus:

This causes the selected window to appear on top of all the other windows. It also "attaches" the keyboard to the selected window. This is the same as using Pop followed by `Keyboard Focus`.

1. Select Pop/Focus from the menu with the mouse. The pointer will appear as a shaded five sided figure.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. The window selected will appear on top of all other windows. Keyboard input will also be directed to the selected window.

Pop/Focus/Resize:

This causes the selected window to appear on the top of the other windows, have the keyboard "attached" to it, and expand in size (to full width and approximately three-quarter height).

1. Select `Pop/Focus/Resize` from the menu with the mouse. The pointer will appear as an up arrow enclosed by four triangles.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. The window selected will:
   - appear on top of all the other windows
   - have the keyboard "attached" to it
   - expand in size

Exit:

Use this selection to leave the menu list without effecting the windows.

Expanded Menu

This section explains how to use the selections provided by the expanded menu (See Figure A-3). This menu is invoked by the System V/386 operating (see the *iSBXT 279 Display Subsystem Installation Guide*).

Figure A-3: Expanded Menu Selections

Window Operations
Pan
Attach Keyboard
Pop to Foreground
Push to Background
Pop and Set Focus
Move Window
Resize Window
Expand Window
Reduce Window
Map Window

Pan:

This allows the contents of a window to be moved. This selection is useful for viewing the contents of a window that has been reduced in size.

1. Select `Pan` from the menu with the mouse. The pointer will appear as a square with four arrows pointing out from each side of the square.
2. Place the pointer within the desired window. Press and hold down any of the mouse buttons.
3. Moving the mouse will cause the contents of the window to move.
4. Release the mouse button when the desired contents are displayed.

Attach Keyboard:

This selection allows keyboard input to be directed to a window.

1. Select `Attach Keyboard` from the menu with the mouse. The pointer will appear as a shaded box.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. Keyboard input will also be directed to the selected window.

Pop To Foreground:

This causes the selected window to appear on top of all other windows. The keyboard may not be "attached " to the window that has been popped. Use the `Attach Keyboard` selection to attach the keyboard to the popped window.

1. Select `Pop To Foreground` from the menu with the mouse. The pointer will change to an up arrow.
2. Place the pointer within the desired window and press any of the mouse buttons.
3. The window selected will appear on top off all other windows.

Push To Background:

This causes the selected window to be placed behind all other windows.

1. Select `Push To Background` from the menu with the mouse. The pointer will change to a down arrow.
2. Place the pointer within the desired window and press any of the mouse buttons.
3. The window selected will be placed behind all other windows.

Pop and Set Focus:

This causes the selected window to appear on top of all the other windows. It also "attaches" the keyboard to the selected window. A quick way of doing this is to position the pointer within the desired window and pressing any of the mouse buttons.

1. Select `Pop and Set Focus` from the menu with the mouse. The pointer will appear as a shaded five sided figure.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. The window selected will appear on top of all other windows. Keyboard input will also be directed to the selected window.

Move Window:

This allows a window to be moved around the screen.

1. Select `Move Window` from the menu with the mouse. The pointer will appear as two crossed (up/down left/right) bidirectional arrows.
2. Position the pointer within the desired window. Press and hold down any of the mouse buttons.
3. Moving the mouse will cause the entire window to move.
4. Release the mouse button when the window is in the desired location.

Resize Window:

This selection causes the size of the window to be changed.

1. Select `Resize Window` from the menu with the mouse. The pointer will appear as two crossed bidirectional arrows, slightly tilted.
2. Position the pointer within the desired window. The pointer should be near one of the four corners of the window. The corner selected will be the part of the window that moves. Press and hold down any of the mouse buttons.
3. Move the mouse. This causes the size of the window to change.
4. Release the mouse button when the window is at the desired size.

NOTE Window contents are not rescaled when the window is resized. The maximum size of a window is either the size of the bitmap in which it is drawn or the size of the screen.

Expand Window:

This causes the selected window to appear on the top of the other windows, have the keyboard "attached" to it, and expand in size (to full width and approximately three-quarter height).

1. Select `Expand Window` from the menu with the mouse. The pointer will appear as an up arrow enclosed by four triangles.
2. Position the pointer within the desired window and press any of the mouse buttons.
3. The window selected will:
   - appear on top of all the other windows
   - have the keyboard "attached" to it
   - expand in size

Reduce Window:

This selection allows you to shrink the size of a window quickly and move it out of the way of other windows.

Select `Reduce Window` from the menu with the mouse. The pointer will appear as a shaded square with an unshaded middle.

1. Position the pointer within the desired window next to one of the four corners. Press and hold down any one of the mouse buttons.
2. Move the mouse. The window will shrink at a very fast rate.
3. Release the mouse button when the window reaches the desired size and position on the screen.

Map Window:

This selection allows a function key (`F1 - F10`) to be assigned to each window. When the function key is pressed (`ALT - Fx`) the corresponding window will appear on top of the other windows and keyboard input will be directed to the window.

1. Select `Map Window` from the menu with the mouse. The pointer will appear as rectangle inside a grid.
2. Position the pointer within the desired window and press any one of the mouse buttons.
3. The pointer will change to the one shown to the lower right.
4. On the keyboard, hold down the `ALT` key and press the function key you want assigned to the window.
5. To use the function keys to select a window, hold down the `ALT` key and press the corresponding function key. The window assigned to that key will appear on top of the other windows and the keyboard will be attached to it.

NOTE

See the *iSBX? 279 Display Subsystem Installation Guide.*

B. MULTIBUS SYSTEMS

B. MULTIBUS SYSTEMS

B Appendix B

Information Unique to MULTIBUS Systems

This appendix provides information needed in creating block device drivers for Intel systems. The ivlab (volume label) structure and MDL (Manufacturer's Defect List) are both described in this appendix.

Volume Label: ivlab Structure

This volume label is required to be on any block device which is to be used as the boot device in a system using Intel standard bootstrap loaders. This includes the standard monitor PROMs in the System 320 or 520. Other bootstrap loaders will have other requirements of the volume label. Some of the fields are defined for use with the Intel RMXR II operating system and are not used by System V/386.

```
struct       ivlab {
     char    v_name[10];        /* volume name, blank padded */
     char    v_flags;           /* flags byte--see below */
     char    v_fdriver;         /* file-driver number */
     ushort  v_gran;            /* volume-gran (bytes) */
     ushort  v_size_l;          /* size (bytes) of volume (low) */
     ushort  v_size_h;          /* size (bytes) of volume (high) */
     ushort  v_maxfnode;        /* max fnode number, 0 in UNIX */
     ushort  v_stfnode_l;       /* start of fnodes, 2 in UNIX, low */
     ushort  v_stfnode_h;       /* start of hnodes, 2 in UNIX, high*/
     ushort  v_szfnode;         /* size of fnode, 32 in UNIX */
     ushort  v_rfnode;          /* root  fnode, 2 in UNIX */
     ushort  v_devgran;         /* sector size (bytes) */
     ushort  v_intl;            /* interleave; 0==> unknown */
     ushort  v_trskew;          /* track skew; 0==> none */
     ushort  v_sysid;           /* OS id for OS that formatted vol.*/
     char    v_sysname[12];     /* OS name, blank filled */
     char    v_dspecial[8];     /* device-special info */
     ushort  v_fsdelta:         /* start of root file system */
     char    v_freespace[4];    /* free space for future use */
};
```

v_name	This is the name of the volume ASCII, right blank-filled.
v_flags	These flags describe some of the characteristics of the physical device. These characteristics are used by the PROM-based bootstrap loader.

The definition of this byte is defined by:

```
#define VF_AUTO        0x01 /* 1==>byte is valid */
#define VF_DENSITY     0x02 /* 0==FM, 1=MFM */
#define VF_SIDES       0x04 /* 1=double-sided */
#define VF_MINI        0x08 /* 0=8, 1=5.25 */
#define VF_NOT_FLOPPY  0x10 /* 0=flop track 0 is 128SD 1=not floppy */
```

v_fdriver	This field is the ID number of the file driver for this volume. This is primarily used by the RMX Operating Systems, and is simply set to UNIX_FD, to be different from RMX.

```
#define  UNIX_FD     6     /* UNIX file-driver number */
```

v_gran	Volume granularity in bytes per sector. This is a logical granularity, primarily used by RMX. In System V/386, this is always set to 1024.
v_size_l, v_size_h	Low and high order bytes of volume size expressed in bytes. Not used by System V/386.
v_maxfnode	This field is the maximum ordinal number of an fnode in RMX. An fnode is the RMX equivalent to an inode. Set to 0, not used by System V/386.
v_stfnode_l, v_stfnode_h	Low and high order bytes of start of fnodes in RMX. Set to 2, not used by System V/386.
v_szfnode	Size of an fnode, in bytes. Set to size of an inode: 32.
v_stfnode	Start of fnodes. Set to root inode: 2.
v_devgran	Device granularity, in bytes per sector. Physical device granularity, determined when device was formatted.

v_intl	Physical device interleave. When set to 0, indicates unknown. Not used in System V/386.
v_trskew	Track skew. When set to 0, indicates unknown. Not used is System V/386.
v_sysid	ID of operating system that formatted volume. Set to 0x0040 for System V/386.
v_sysname	Name of operating system that formatted volume. Set to *System V: for System V/386. Twelve ASCII characters, right blank-filled.
v_dspecial	Eight bytes that contain device-specific information in the form of a driver table entry. The format of the data is defined by the driver include file (`usr/include/sys/i214.h`) and is limited to 8 bytes. The format is controller dependent.
dr_nalt	This field changes to a density flag if the volume label is on a floppy device. *dr_nalt* set to 0 indicates a single-density, FM-format floppy, and *dr_nalt* set to 1 indicates a double-density, MFM-format floppy.
v_fsdelta	The absolute physical sector number of the start of the root file system.
v_freespace	Four bytes of unused space.

Manufacturer's Defect List: MDL Structure

The MDL is for disk drives using the ST506 interface that are supported by the MULTIBUS II 186/224A or the MULTIBUS 214 or 221 controllers, for example.

The ST506 MDL is written in the next to the last cylinder on four different tracks using a bytes/sector value of 128, 256, 512, and 1024. The track assignments within the cylinder are as follows:

128 bytes/sec	Last track in cylinder
256 bytes/sec	Last track - 1
512 bytes/sec	Last track - 2
1024 bytes/sec	Last track - 3

Each track contains four copies of the MDL starting at the beginning of the track and spaced every 2K-bytes (i.e., the byte offset within the track of each copy is 0K-bytes, 2K-bytes, 4K-bytes, and 8K-bytes).

```
#define BBH506MAXDFCTS     255    /* max # of ST506 defects */
struct st506mdl {                 /* ST506 MDL */
      struct st506hdr     header;
      struct st506defect  defects [BBH506MAXDFCTS];
      };

struct st506hdr {                 /* ST506 header information */
      unsigned short      bb_valid;
      unsigned short      bb_num;
      };

struct st506defect {              /* ST506 individual defect info */
      unsigned short      be_cyl;
      unsigned char       be_surface;
      unsigned char       be_reserved;
      };
```

C. RELATED PUBLICATIONS

C. RELATED PUBLICATIONS

C Appendix C

Related Publications

The publications listed below provide additional information about the System V/386 operating system.

The C Programmer's Handbook, Bell Labs/M. I. Bolsky
The UNIX System User's Handbook, Bell Labs/M. I. Bolsky
The Vi User's Handbook, Bell Labs/M. I. Bolsky
UNIX System Software Readings, AT&T UNIX PACIFIC
UNIX System Readings and Applications, Volume I, Bell Labs
UNIX System Readings and Applications, Volume II, Bell Labs
UNIX System V/386 Release 4.0 Manual Set, AT&T (published by Prentice Hall)

Intel System How To Use The Peripheral Controller Interface (PCI) Server
Intel System V/386 MULTIBUS Installation and Configuration
Intel System V/386 MULTIBUS II Transport-Application Interface User's Guide
MULTIBUS II Transport Protocol Specification and Designer's Guide
System V/386 Device Driver Interface/Driver-Kernel Interface Reference Manual
MULTIBUS II Interconnect Interface Specification
MULTIBUS II Message Passing Coprocessor External Specification
MULTIBUS II MPC User's Guide

320–705

UNIX
PRESS

A Prentice Hall Title

ISBN 0-13-957481-6