UNIX® SYSTEM V/386
RELEASE 4
MULTIBUS® Reference Manual

UNIX® System V

UNIX Software Operation
ORDERING INFORMATION

UNIX® SYSTEM V, RELEASE 4 DOCUMENTATION

To order single copies of UNIX® SYSTEM V, Release 4 documentation, please call (201) 767-5937.

ATTENTION DOCUMENTATION MANAGERS AND TRAINING DIRECTORS:
For bulk purchases in excess of 30 copies please write to:
Corporate Sales
Prentice Hall
Englewood Cliffs, N.J. 07632
Or call: (201) 592-2498

ATTENTION GOVERNMENT CUSTOMERS: For GSA and other pricing information please call (201) 767-5994.

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
AT&T UNIX® System V Release 4

General Use and System Administration

*UNIX® System V/386 Release 4 PC-Interface Administrator's Guide
*UNIX® System V/386 Release 4 Product Overview and Master Index
*UNIX® System V/386 Release 4 MULTIBUS® Installation and Configuration Guide
*UNIX® System V/386 Release 4 Mouse Driver Administrator's Guide
UNIX® System V Release 4 User's Guide
UNIX® System V Release 4 System Administrator’s Guide

General Programmer's Series

UNIX® System V Release 4 Programmer's Guide: Character User Interface (FMLI and ETI)

System Programmer's Series


Migration Series

UNIX® System V Release 4 ANSI C Transition Guide
UNIX® System V Release 4 BSD/XENIX® Compatibility Guide
*UNIX® System V/386 Release 4 Migration Guide

Graphics Series


*386 specific titles
Available from Prentice Hall
Contents

1 Manual Overview
   Manual Overview 1-1

2 MULTIBUS Commands
   MULTIBUS I and MULTIBUS II Commands 2-1

3 Functions And Libraries
   Interconnect-Space Application Interface 3-1

4 Intel's System V/386 File Formats
   Intel's System V/386 File Formats 4-1

5 Device Information
   Disk And Tape Device Drivers 5-1

6 Device Drivers
   Device Drivers 6-1

Table of Contents i
Table of Contents

7  System Maintenance Commands
    Maintenance Commands 7-1

8  Static Bad Block Handling
    Overview 8-1
    Architecture 8-2

Appendix A
    Window Interface 1

Appendix B
    Information Unique to MULTIBUS Systems 1

Appendix C
    Related Publications 1

MULTIBUS Reference Manual
Figures and Tables

Figure 1: An Example of Windows Displayed on the SYP 520 2
Figure 2: Basic Menu Selections 4
Figure 3: Expanded Menu Selections 7
devicedriver i224a iSBC
device driver i224atp iSBC
ccontroller device driver i4(1M)
driver i530 iSBC
device driver i214 iSBCR
device driver i214tp iSBC
Data Link driver for iNA961 Release
i8251 console terminal/iSBXT
driver i354 iSBX
device driver i258 iSBC
device driver i258tp iSBC
i546 iSBC
module formats download loads
general formats download loads 8086
/services, icsgetrec utilities to
d258 i82258
cperms set
cpout copy out file
script devicedriver atcs
driver iasy
controller script device driver
on the communication/ ccibind
/reads interconnect registers of the
in the interconnect space of a
into interconnect registers of the
bootserver starts a MULTIBUS II
boot service daemon
sgib install MULTIBUS
bps
processor/ initbp initializes the
driver
board and returns the values in the
interface device driver
communications/ ccisrvinfo
subchannel on the communications/
a line on the communication/
with a subchannel on a line on the/
program on the communications/
information on the communications/
the communications controller
communications controller
program on to the communications/
on the communications controller
on the communications controller
line to a new host on the/
from a line on the communications/
i354 iSBX 354 dual
186/224A peripheral controller disk ...................................... i224a(1M)
186/224A peripheral controller tape ...................................... i224atp(1M)
186/410 serial communications ........................................... i4(1M)
186/530 Ethernet controller device ...................................... i530(1M)
214 peripheral controller disk ........................................... i2(1M)
214 peripheral controller tape ........................................... i2(1M)
3.0 edlina External ...................................................... edlina(lM)
351 driver ...................................................................... i825(lM)
354 dual channel serial-port device ...................................... i354(1M)
386/258 peripheral controller .............................................. i258(1M)
386/258 peripheral controller tape ...................................... i258tp(1M)
546 multi-port serial controller ........................................... i546(1M)
8086 absolute library and object ........................................... download(lM)
absolute library and object module ..................................... download(lM)
access MULTIBUS II interconnect/ icsrd(lM)
ADMA device driver ...................................................... d258(1M)
and/or verify permissions on files ....................................... ckperms(lM)
archives to multiple volumes ............................................. cpout(lM)
asynchronous terminal controller ...................................... atcs(lM)
asynchronous (terminal) device ......................................... iasy(lM)
ats asynchronous terminal .............................................. atcs(lM)
binds a line discipline to a line ....................................... ccibind(lM)
board and returns the values in the/ ics_read(lM)
board /finds a specific record ......................................... ics_find_rec(lM)
board ics_write writes ............................................... ics_write(lM)
boot service daemon ...................................................... bootserver(lM)
bootstrap loaders ......................................................... sgib(lM)
Bootstrap Parameter String driver ...................................... bps(lM)
bootstrap parameter string on the ................................. initbp(lM)
bps Bootstrap Parameter String ......................................... bps(lM)
buffer /registers of the ............................................... ics_read(lM)
cci communications controller ......................................... cci(lM)
CCI server information on the ........................................ ccisrvinfo(lM)
ccattach creates a connection to a ................................. ccattach(lM)
ccbind binds a line discipline to ..................................... ccbind(lM)
ccdetach disassociates this host ...................................... ccdetach(lM)
ccfree frees a line discipline ......................................... ccfree(lM)
ccidinfo line discipline ................................................. ccidinfo(lM)
cclidiscipline list on ................................................. ccidiscipline(lM)
cclidlinfo line information on the .................................. ccidlinfo(lM)
cclload load a line discipline ........................................ ccclload(lM)
ccisrvinfo CCI server information ................................... ccisrvinfo(lM)
ccsubchannel information ............................................. ccsubchannel(lM)
ccswswitch switch a subchannel on a ......................... ccswswitch(lM)
cciunbind unbinds a line discipline ................................ cciunbind(lM)
channel serial-port device driver .................................... i354(1M)
communications/cciattach Creates a
   a connection to a subchannel on the
   with a subchannel on a line on the
   a line discipline on the
   /line discipline information on the
   /line discipline list on the
cciinfo line information on the
   a line discipline program on to the
   /CCI server information on the
   /subchannel information on the
   /on a line to a new host on the
line discipline from a line on the
driver i410 iSBC 186/410 serial
device driver cci
   a line to a new host on the
file format
config MULTIBUS II
communications/cciattach Creates a
   console
   i8251
rci debug
   a subchannel on the communications
to a line on the communication
   on a line on the communications
   program on the communications
   information on the communications
   list on the communications
   information on the communications
   program on to the communications
   information on the communications
to a new host on the communications
   from a line on the communications
i258 iSBC 386/258 peripheral
iSBC 186/410 serial communications
i530 iSBC 186/530 Ethernet
i214 iSBCR 214 peripheral
i224a iSBC 186/224A peripheral
on an ethernet node9s ethernet
resets and reloads the ethernet
i546 iSBC 546 multi-port serial
c costume
ats asynchronous terminal
i214tp iSBC 214 peripheral
controller tape device driver ................................. i224atp(1M)
controller tape device driver ................................. i258tp(1M)
controller /switches a line .................................... ttyswitch(IM)
Convert transport code ......................................... mpscnv(IM)
copy out file archives to multiple .......................... cpout(IM)
copy out file archives to ....................................... cpout(IM)
Creates a connection to a ................................. cciattach(IM)
d258 i82258 ADMA device driver .......................... d258(IM)
daemon bootserver .............................................. bootserver(IM)
Data Link driver for iNA961 Release ...................... edlina(IM)
dbon sets target for front panel .............................. dbon(IM)
debug console/rci protocol driver ........................... rci(IM)
defect list mdl read and ......................................... mdl(IM)
delivery dbon ..................................................... dbon(IM)
device driver atcs asynchronous ......................... atcs(IM)
device driver cci ................................................... cci(IM)
device driver console ............................................. console(IM)
device driver d258(IM)
device driver dma(IM)
device driver i214(ISBCR) ....................................... i2(IM)
device driver i214tp ............................................... i2(IM)
device driver i224a(ISBC) ...................................... i224a(IM)
device driver i224atp ISBC ................................. i224atp(IM)
device driver i258 .................................................. i258(IM)
device driver i258tp ISBC ...................................... i258tp(IM)
device driver i354 .................................................. i354(IM)
device driver i410 ISBC 186/410 ....................... i4(IM)
device driver i530 .................................................. i530(IM)
device driver iasy(IM)
device driver ics(IM)
device driver mpc(IM)
device driver mps(IM)
device driver ots(IM)
device driver fddd(IM)
device driver hddd(IM)
device driver tdd(IM)
disassociates this host with a .............................. cciattach(IM)
discipline from a line on the/ ............................... cciunbind(IM)
discipline information on the ......................... cciunbind(IM)
discipline program on the ......................... cciunload(IM)
discipline program on to the ......................... cciunload(IM)
discipline to a line on the ............................... cciunload(IM)
disk device driver i214 ....................................... i2(IM)
disk device driver i224a ..................................... i224a(IM)
disk device drivers ........................................... fddd(IM)
disk device drivers ........................................... hddd(IM)
MULTIBUS Reference Manual
sgib install MULTIBUS bootstrap loaders .......................... sgib(1M)
interconnect registers of the board .............................. ics_read(1M)
ics and returns the/ ics_read reads
ics_write writes into
ics interconnect registers of the board .......................... ics_write(1M)
interconnect Space device driver .............................. ics(1M)
ics_interconnect space /icsgetrec .............................. icsrd(1M)
ics_interconnect space of a board .............................. ics_find_rec(1M)
interface device driver ................................. cci(1M)
iSBC 186/224A peripheral controller .......................... i224a(1M)
iSBC 186/224A peripheral controller .......................... i224atp(1M)
iSBC 186/410 serial communications ........................... i4(1M)
iSBC 186/530 Ethernet controller ............................. i530(1M)
iSBC 214 peripheral controller tape ........................... i2(1M)
iSBC 386/258 peripheral controller ........................... i258(1M)
iSBC 386/258 peripheral controller ........................... i258tp(1M)
iSBC 546 multi-port serial .............................. i546(1M)
iSBCR 214 peripheral controller .............................. i2(1M)
iSBX 354 dual channel serial-port ............................. i354(1M)
hdformat low level hard disk formatter .................. hdformat(1M)
download loads 8086 absolute library and object module formats download(1M)
communications/ ccunbind unbinds a line discipline from a line on the communication/ ccunbind(lM)
communications/ ccildinfo line discipline information on the communication/ ccildinfo(lM)
communications/ ccildlist line discipline program on the communication/ ccildlist(lM)
communications/ ccifree frees a line discipline program on to the communication/ ccifree(lM)
communications/ cciload load a line discipline program on to the communication/ cciload(lM)
communications/ cciswitch switch a line discipline to a new host on the communication/ cciswitch(lM)
communications/ ccsbind binds a line discipline to the communications/ ccsbind(lM)
communications/ ccldlist line discipline to the communications/ ccldlist(lM)
sgib install MULTIBUS bootstrap loaders .......................... sgib(1M)
object module formats download loads 8086 absolute library and object module formats download(1M)
hdformat low level hard disk formatter .................. hdformat(1M)
mdl read and optionally write the manufacturer’s defect list mdl(1M)
list on the communications/ cclist(lM)
mdl read and optionally write the manufacturer’s defect list mdl(1M)
load a line discipline program on cclist(lM)
loaders sgib(1M)
loads 8086 absolute library and object module formats download loads 8086 absolute library and object module formats download(1M)
low level hard disk formatter .................. hdformat(1M)
manufacturer’s defect list mdl(1M)
mdl read and optionally write the manufacturer’s defect list mdl(1M)
message delivery dbon(1M)
Message Passing Space device driver mps(1M)
module formats download loads download loads mps(1M)
MPC device driver mpc(1M)
mpc MPC device driver mpc(1M)
mps Message Passing Space device mps(1M)
mpsconvert Convert transport code mpscnv(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)
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 System V/386 OSI Transport Service ............ ots(1M)
driver ots System V/386 OSI Transport Service (ots) device .......... ots(1M)
driver ots System V/386 OSI Transport Service (ots) device driver ...... ots(1M)
driver ots System V/386 OSI Transport Service panel message delivery .... dbon(1M)
bps Bootstrap Parameter String driver ....................................... bps(1M)
initbp initializes the bootstrap parameter string on the processor/ ...................... initbp(1M)
mp5 Message Passing Space device driver .................................. mp5(1M)
driver 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 ........... i2(1M)
driver i214tp iSBC 214 peripheral controller tape device ............... i2(1M)
driver i224atp iSBC 186/224A peripheral controller tape device .......... i2(1M)
driver i258tp iSBC 386/258 peripheral controller tape device .......... i2(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 controller board returns the/ ics_read a/ ics_find_rec finds a specific the/ ics_read reads interconnect ics_write writes into interconnect Data Link driver for iNA961 enetload resets and given slot controller enetload ramd MULTIBUS RAM disk driver ............................................. ramd(1M)
rci debug console/rci protocol ............................................. rci(1M)
read and optionally write the manufacturer’s ............................. mdl(1M)
reads interconnect registers of the record in the interconnect space of registers of the board and returns registers of the board .............................................. ics_read(1M) Release 3.0 edlina External ................................................... edlina(1M)
reloads the ethernet controller ............................................... enetload(1M)
reset resets the processor in a ............................................... reset(1M)
resets and reloads the ethernet ............................................. enetload(1M)
slot reset
/set registers of the board and
asynchronous terminal controller
device driver i410 iSBC 186/410
i546 iSBC 546 multi-port
i354 iSBX 354 dual channel
communications /ccisrvinfo CCI
starts a MULTIBUS II boot
ots System V/386 OSI Transport
files ckperms
delivery dbon
loaders
on the processor in the given
resets the processor in a given
ics Interconnect
mps Message Passing
 Space device driver ................................................................. ics(1M)
to access MULTIBUS II interconnect
specific record in the interconnect
space /icsslot, icsgetrec finds a
daemon bootserver
bps Bootstrap Parameter
/initializes the bootstrap parameter
communications /ccisubinfo
/disassociates this host with a
on the /cciswitch switch a
cclattach Creates a connection to a
new host on the /cciswitch
the communications /ttyswitch
(ots) device driver ots
iSBC 214 peripheral controller
iSBC 186/224A peripheral controller
iSBC 386/258 peripheral controller
tdd MULTIBUS
delivery dbon sets
driver atcs asynchronous
iasy asynchronous
i8251 console
mpscnv Convert
driver ots System V/386 OSI
host on the communications/
line on the /ccunbind
icsrd, icswr, icsslot, icsgetrec
device driver ots System
of the board and returns the
ckperms set and/or
resets the processor in a given
returns the values in the buffer
rtc MULTIBUS clock driver
script device driver atcs
serial communications controller
serial controller
serial-port device driver
server information on the
service daemon bootserver
Service (ots) device driver
set and/or verify permissions on
sets target for front panel message
sgib install MULTIBUS bootstrap
glb(lM)
slot /bootstrap parameter string
slot reset
Space device driver
Space device driver
space /icsslot, icsgetrec utilities
specific record in the interconnect
specific record in the interconnect
start a MULTIBUS II boot service
String driver
String driver
string on the processor in the/
subchannel information on the
subchannel on a line on the/
subchannel on a line on a new host
subchannel on the communications/
switch a subchannel on a line to a
switches a line to a new host on
System V/386 OSI Transport Service
tape device driver
tape device driver
tape device driver
tape device drivers
tape device drivers
target for front panel message
tdd MULTIBUS tape device drivers
tdd MULTIBUS tape device drivers
tdd MULTIBUS tape device drivers
(terminal) device driver
(terminal) device driver
(terminal) device driver
transport code
Transport Service (ots) device
ttyswitch switches a line to a new
unbinds a line discipline from a
utilities to access MULTIBUS II/
V/386 OSI Transport Service (ots)
values in the buffer /registers
verify permissions on files
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 Manual Overview

Manual Overview

Organization 1-2
Notational Conventions 1-2
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 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
Appendix B.
    Information Unique to MULTIBUS Systems

Appendix C.
    Related Publications

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 MULTIBUS Commands

MULTIBUS I and MULTIBUS II Commands 2-1
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
- hiformat
- mdl
- sgib

Commands that apply only to MULTIBUS II are:

- bootserver
- cciattach
- ccibind
- ccidetach
- ccifree
- ccildinfo
- ccildlist
- ccilinfo
- cciload

- ccisrvinfo
- ccisubinfo
- cciswitch
- cciunbind
- download
- enetinfo
- enetload
- mpscnv
- ttyswitch
NAME

ckperms - set and/or verify permissions on files

SYNOPSIS

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
[-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 pro-
cessed 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(1ref)).

   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(1ref), mdl(1ref)
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 informa-
  tion 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 hdfor-
  mat 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 amount 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 addi-
  tional 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

[-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
cclattach - Creates a connection to a subchannel on the communications controller.

SYNOPSIS
/usr/lib/ccl/cclattach [-p portid] slot-ID line-number sub-channel

DESCRIPTION

cclattach 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/ccl/cclattach

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(liref) 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(liref) 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(liref), ccildlist(liref), ccilinfo(liref),
ccisubinfo(liref), ccildinfo(liref), ccifree(liref),
cciunbind(liref), cciload(liref), cciattach(liref),
ccidetach(liref), cciswitch(liref), cci(liref)
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(liref) 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 0xffff.

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(liref) command. The subchannel can range from 0 to MAX_SUBCHANNELS - 1, where MAX_SUBCHANNELS is returned in the ccibind(liref) command. The communications controller is selected by specifying its MULTIBUS II IPSB slot-ID.

FILES
/usr/lib/cci/ccidetach

SEE ALSO
ccisrvinfo(liref), ccildlist(liref), ccilinfo(liref), ccisubinfo(liref), ccildinfo(liref), ccifree(liref), cciunbind(liref), cciload(liref), cciattach(liref), cciswitch(liref), cci(7liref)
ccifree(1M)

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 previ­
ously downloaded to the communications controller using the cciload(liref) 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(liref)
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
the 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(liref), ccilinfo(liref), cciunbind(liref),
ccisubinfo(liref), ccidinfo(liref), cciattach(liref),
ccunbind(liref), cciload(liref), ccidetach(liref),
cciswitch(liref), cci(7iref)
NAME
cclidinfo - line discipline information on the communications controller.

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

DESCRIPTION
cclidinfo 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(liref), ccildlist(liref), ccilinfo(liref),
ccisubinfo(liref), cciload(liref), ccibind(liref),
cciunbind(liref), ccifree(liref), cciattach(liref),
ccidetach(liref), cciswitch(liref)
ccildlist (1M) ccildlist (1M)

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(liref), ccildinfo(liref), ccilinfo(liref),
ccisubinfo(liref), cciload(liref), ccibind(liref),
cciunbind(liref), ccifree(liref), cciattach(liref),
ccidetach(liref), cciswitch(liref)
ccilinfo(1M)

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), ccilist(1iref), ccilinfo(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(liref) or the ccildinfo(liref) 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(liref), ccildlist(liref), ccilinfo(liref),
ccisubinfo(liref), ccildinfo(liref), ccibind(liref),
cciunbind(liref), ccfree(liref), cciattach(liref),
ccidetach(liref), cciswitch(liref)
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 con­
troller hosting the CCI server is selected by specifying its MULTIBUS II iPSB
slot-ID.

FILES
/usr/lib/cci/ccisrvinfo

SEE ALSO
ccildlist(liref), ccildinfo(liref), ccilinfo(liref),
ccisubinfo(liref), ccisload(liref), ccisbind(liref),
ccisunbind(liref), ccisfree(liref), ccisattach(liref),
ccisdetach(liref), ccisswitch(liref)
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(liref), ccildlist(liref), ccildinfo(liref), ccilinfo(liref), ccilload(liref), ccibind(liref), ccunbind(liref), ccifree(liref), cciattach(liref), ccidetach(liref), cciswitch(liref)
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(liref) 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(liref) command. The sub-channel can range from 0 to MAX_SUBCHANNELS - 1, where MAX_SUBCHANNELS is returned in the ccibind(liref) 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(liref), ccildlist(liref), ccilinfo(liref),
 ccisubinfo(liref), ccildinfo(liref), ccifree(liref),
 cciunbind(liref), cciload(liref), ccidetach(liref),
 cciattach(liref)
NAME
cclunbind - unbinds a line discipline from a line on the communications controller.

SYNOPSIS
/usr/lib/cci/cclunbind [-p portid] slot-ID line-number

DESCRIPTION
cclunbind deletes the association established between a line, a line discipline and this host by an earlier ccibind (liref) 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(liref) command. The communications controller hosting the CCI server is selected by specifying its MULTIBUS II iPSB slot-ID.

FILES
/usr/lib/cci/cclunbind

SEE ALSO
ccisrvinfo(liref), ccildlist(liref), ccilinfo(liref),
ccisubinfo(liref), ccildinfo(liref), ccifree(liref),
ccibind(liref), cciload(liref), cciattach(liref),
cclidetach(liref), cciswitch(liref)
download(1M)

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:
-bn Set the temporary buffer size to n bytes instead of the default 32,000 bytes.
-tn Set the number of milliseconds to wait between transport messages used to transfer the image to the target. The default is 10.
-pn 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 sys-
    tem.
- `-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

mpsncv - Convert transport code

DESCRIPTION

The `mpsncv` 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.

<table>
<thead>
<tr>
<th>Release 3.2 Identifier</th>
<th>Release 4.0 Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPcancel</td>
<td>mps_AMPcancel</td>
</tr>
<tr>
<td>AMPreceive</td>
<td>mps_AMPreceive</td>
</tr>
<tr>
<td>AMPreceive_frag</td>
<td>mps_AMPreceive_frag</td>
</tr>
<tr>
<td>AMPsend</td>
<td>mps_AMPsend</td>
</tr>
<tr>
<td>AMPsend_data</td>
<td>mps_AMPsend_data</td>
</tr>
<tr>
<td>AMPsend_reply</td>
<td>mps_AMPsend_reply</td>
</tr>
<tr>
<td>AMPsend_rsvp</td>
<td>mps_AMPsend_rsvp</td>
</tr>
<tr>
<td>BLKPRIO</td>
<td>MPS_BLKPRIO</td>
</tr>
<tr>
<td>BOARD_SPECIFIC_REC</td>
<td>ICS_BOARD_SPECIFIC_REC</td>
</tr>
<tr>
<td>CLKPRIO</td>
<td>MPS_CLKPRIO</td>
</tr>
<tr>
<td>EOT_TYPE</td>
<td>ICS_EOL_TYPE</td>
</tr>
<tr>
<td>FP_PORT</td>
<td>MPS_FP_PORT</td>
</tr>
<tr>
<td>HI</td>
<td>ICS_HI</td>
</tr>
<tr>
<td>HIB</td>
<td>MPS_HIB</td>
</tr>
<tr>
<td>HOST_ID_TYPE</td>
<td>ICS_HOST_ID_TYPE</td>
</tr>
<tr>
<td>ICBistDataIn</td>
<td>ICS_BistDataIn</td>
</tr>
<tr>
<td>ICBistDataOut</td>
<td>ICS_BistDataOut</td>
</tr>
<tr>
<td>ICBistMasterStatus</td>
<td>ICS_BistMasterStatus</td>
</tr>
<tr>
<td>ICBistSlaveStatus</td>
<td>ICS_BistSlaveStatus</td>
</tr>
<tr>
<td>ICBistSupportLevel</td>
<td>ICS_BistSupportLevel</td>
</tr>
<tr>
<td>ICBistTestID</td>
<td>ICS_BistTestID</td>
</tr>
<tr>
<td>IClassID</td>
<td>ICS_ClassID</td>
</tr>
<tr>
<td>ICGenericControl</td>
<td>ICS_GenericControl</td>
</tr>
<tr>
<td>ICGenericStatus</td>
<td>ICS_GenericStatus</td>
</tr>
<tr>
<td>ICHardwareTestRev</td>
<td>ICS_HardwareTestRev</td>
</tr>
<tr>
<td>ICNMIEnable</td>
<td>ICS_NMIEnable</td>
</tr>
<tr>
<td>ICPProductCode</td>
<td>ICS_ProductCode</td>
</tr>
<tr>
<td>ICPProgramTableIndex</td>
<td>ICS_ProgramTableIndex</td>
</tr>
<tr>
<td>ICVendorIDH</td>
<td>ICS_VendorIDH</td>
</tr>
<tr>
<td>ICVendorIDL</td>
<td>ICS_VendorIDL</td>
</tr>
<tr>
<td>IC_DEBUG_MASK</td>
<td>ICS_DEBUG_MASK</td>
</tr>
<tr>
<td>IC_DEBUG_NMI</td>
<td>ICS_DEBUG_NMI</td>
</tr>
<tr>
<td>IC_ENABLE_NMI</td>
<td>ICS_ENABLE_NMI</td>
</tr>
<tr>
<td>IC_PARITY_MASK</td>
<td>ICS_PARITY_MASK</td>
</tr>
<tr>
<td>IC</td>
<td>ICS</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>IC_PARITY_OFFSET</td>
<td>ICS_PARITY_OFFSET</td>
</tr>
<tr>
<td>IC_SWMNI_MASK</td>
<td>ICS_SWMNI_MASK</td>
</tr>
<tr>
<td>IC_SW_NMI</td>
<td>ICS_SW_NMI</td>
</tr>
<tr>
<td>LO</td>
<td>ICS_LO</td>
</tr>
<tr>
<td>LOB</td>
<td>MPS_LOB</td>
</tr>
<tr>
<td>MAXMSGSZ</td>
<td>MPS_MAXMSGSZ</td>
</tr>
<tr>
<td>MAX_REG</td>
<td>ICS_MAX_REG</td>
</tr>
<tr>
<td>MAX_SLOT</td>
<td>ICS_MAX_SLOT</td>
</tr>
<tr>
<td>MB2_TPDT</td>
<td>MPS_MB2_TPDT</td>
</tr>
<tr>
<td>MG_BGDC</td>
<td>MPS_MG_BGDC</td>
</tr>
<tr>
<td>MG_BGLI</td>
<td>MPS_MG_BGLI</td>
</tr>
<tr>
<td>MG_BGLL</td>
<td>MPS_MG_BGLL</td>
</tr>
<tr>
<td>MG_BGOVHD</td>
<td>MPS_MG_BGOVHD</td>
</tr>
<tr>
<td>MG_BGRANT</td>
<td>MPS_MG_BGRANT</td>
</tr>
<tr>
<td>MG_BJLI</td>
<td>MPS_MG_BJLI</td>
</tr>
<tr>
<td>MG_BJOVHD</td>
<td>MPS_MG_BJOVHD</td>
</tr>
<tr>
<td>MG_BRDCST</td>
<td>MPS_MG_BRDCST</td>
</tr>
<tr>
<td>MG_BRDP</td>
<td>MPS_MG_BRDP</td>
</tr>
<tr>
<td>MG_BREQ</td>
<td>MPS_MG_BREQ</td>
</tr>
<tr>
<td>MG_BREQ</td>
<td>MPS_MG_BREQ</td>
</tr>
<tr>
<td>MG_BRLI</td>
<td>MPS_MG_BRLI</td>
</tr>
<tr>
<td>MG_BRML</td>
<td>MPS_MG_BRML</td>
</tr>
<tr>
<td>MG_BROVHD</td>
<td>MPS_MG_BROVHD</td>
</tr>
<tr>
<td>MG_BRP1</td>
<td>MPS_MG_BRP1</td>
</tr>
<tr>
<td>MG_BRSP</td>
<td>MPS_MG_BRSP</td>
</tr>
<tr>
<td>MG_BRTC</td>
<td>MPS_MG_BRTC</td>
</tr>
<tr>
<td>MG_BRTI</td>
<td>MPS_MG_BRTI</td>
</tr>
<tr>
<td>MG_BRTI</td>
<td>MPS_MG_BRTI</td>
</tr>
<tr>
<td>MG_BRUD</td>
<td>MPS_MG_BRUD</td>
</tr>
<tr>
<td>MG_DA</td>
<td>MPS_MG_DA</td>
</tr>
<tr>
<td>MG_DFBIND</td>
<td>MPS_MG_DFBIND</td>
</tr>
<tr>
<td>MG_DONE</td>
<td>MPS_MG_DONE</td>
</tr>
<tr>
<td>MG_DUTY1C</td>
<td>MPS_MG_DUTY1C</td>
</tr>
<tr>
<td>MG_DUTY2C</td>
<td>MPS_MG_DUTY2C</td>
</tr>
<tr>
<td>MG_ESOL</td>
<td>MPS_MG_ESOL</td>
</tr>
<tr>
<td>MG_MT</td>
<td>MPS_MG_MT</td>
</tr>
<tr>
<td>MG_NFDP</td>
<td>MPS_MG_NFDP</td>
</tr>
<tr>
<td>MG_NFLL</td>
<td>MPS_MG_NFLL</td>
</tr>
<tr>
<td>MG_NFOVHD</td>
<td>MPS_MG_NFOVHD</td>
</tr>
<tr>
<td>MG_NFPI</td>
<td>MPS_MG_NFPI</td>
</tr>
<tr>
<td>MG_NFSP</td>
<td>MPS_MG_NFSP</td>
</tr>
<tr>
<td>MG_NFTC</td>
<td>MPS_MG_NFTC</td>
</tr>
<tr>
<td>MG_NFTI</td>
<td>MPS_MG_NFTI</td>
</tr>
<tr>
<td>MG_REQ</td>
<td>MPS_MG_REQ</td>
</tr>
<tr>
<td>MG_RES</td>
<td>MPS_MG_RES</td>
</tr>
<tr>
<td>MG_RI</td>
<td>MPS_MG_RI</td>
</tr>
<tr>
<td>MG_RQMF</td>
<td>MPS_MG_RQMF</td>
</tr>
<tr>
<td>MG_RQNRF</td>
<td>MPS_MG_RQNRF</td>
</tr>
<tr>
<td>MG_RQSF</td>
<td>MPS_MG_RQSF</td>
</tr>
<tr>
<td>MG_RQXF</td>
<td>MPS_MG_RQXF</td>
</tr>
</tbody>
</table>
MG_RRMSK
MG_RRTMSK
MG_RSCN
MG_RSET
MG_RSNE
MG_SA
MG_TERR
MG_UMDP
MG_UMOVHD
MG_UMPI
MG_UMSP
MG_UMTC
MG_UMTI
MG_UMUD
MG_UNSOL
MY_SLOT_ID
NRMPRIO
PARITY_CONTROL_TYPE
PC16_CONFIG_OFFSET
PC16_STATUS_OFFSET
PSB_CONTROL_TYPE
PSB_SLOT_ID_REG
READ
SLOT_ID_OFFSET
SRLPRIO
WRITE
agent_cnp
buf count
close_chan
cpunum
datbuf_t
db_join
dbuf_breakup
fetch_db
fetch_mb
find_port
find_transaction
free_dbuf
free_mb
free_msgbuf
free_tid
get_dbuf
get_mb
get_msgbuf

MPS_MG_RRMSK
MPS_MG_RRTMSK
MPS_MG_RSCN
MPS_MG_RSET
MPS_MG_RSNE
MPS_MG_SA
MPS_MG_TERR
MPS_MG_UMDP
MPS_MG_UMOVHD
MPS_MG_UMPI
MPS_MG_UMSP
MPS_MG_UMTC
MPS_MG_UMTI
MPS_MG_UMUD
MPS_MG_UNSOL
ICS_MY_SLOT_ID
MPS_NRMPRIO
ICS_PARITY_CONTROL_TYPE
ICS_PCI6_CONFIG_OFFSET
ICS_PCI6_STATUS_OFFSET
ICS_PSB_CONTROL_TYPE
ICS_PSB_SLOT_ID_REG
ICS_READ
ICS_SLOT_ID_OFFSET
MPS_SRLPRIO
ICS_WRITE
ics_agent_cnp
mps_buf_count
mps_close_chan
ics_cpunum()
mps_datbuf_t
dma_buf_join
dma_buf_breakup
mps_fetch_db
mps_fetch_mb
mps_find_port
mps_find_transaction
mps_free_dmabuf
mps_free_mb
mps_free_msgbuf
mps_free_tid
mps_get_dmabuf
mps_get_mb
mps_get_msgbuf
get_soldata  
get_tid  
get_unsoldata  
handle_err  
init_dmacb  
lhid  
mk_bgrant  
mk_brdcst  
mk_breject  
mk_mb2socid  
mk_mb2soctohid  
mk_mb2soctopid  
mk_snf  
mk_sol  
mk_solrply  
mk_unsol  
mk_unsolrply  
mpc_copyright  
mpc_send  
mpc_start  
msg_comp  
msg_dispatch  
msg_getbrlen  
msg_getdstmid  
msg_getdtpid  
msg_getfraglen  
mgs_getlsnid  
mgs_getmsgtyp  
mgs_getprotid  
mgs_getreqid  
mgs_getrid  
mgs_getsrcmid  
mgs_getsrcpid  
mgs_gettransct1  
mgs_gettransctld  
mgs_getudp  
mgs_iscancel  
mgs_iscompletion  
mgs_iserror  
mgs_isrct  
mgs_isreq  
mgs_iterc  
mgs_setbglen  
mgs_setbrlen  
mgs_setcancel  
mgs_setduty  
mgs_setid  
mgs_setrcpid  

mps_get_soldata  
mps_get_tid  
mps_get_unsoldata  
mps_handle_err  
impc_init_dmacb  
mps_lhid()  
mps_mks_bgrant  
mps_mks_brdcst  
mps_mks_breject  
mps_mks_mb2socid  
mps_mks_mb2soctohid  
mps_mks_mb2soctopid  
mps_mks_snf  
mps_mks_sol  
mps_mks_solrply  
mps_mks_unsol  
mps_mks_unsolrply  
impc_copyright  
impc_send  
impc_start  
mps_msg_comp  
mps_msg_dispatch  
mps_msg_getbrlen  
mps_msg_getdstmid  
mps_msg_getdtpid  
mps_msg_getfraglen  
mps_msg_getlsnid  
mps_msg_getmsgtyp  
mps_msg_getprotid  
mps_msg_getreqid  
mps_msg_getrid  
mps_msg_getsrcmid  
mps_msg_getsrcpid  
mps_msg_gettransct1  
mps_msg_gettransctld  
mps_msg_getudp  
mps_msg_iscancel  
mps_msg_iscompletion  
mps_msg_iserror  
mps_msg_isreq  
mps_msg_iterc  
mps_msg_setbglen  
mps_msg_setbrlen  
mps_msg_setcancel  
mps_msg_setduty  
mps_msg_setid  
mps_msg_setrcpid
SEE ALSO

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
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.
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/librrt>2.a
```
ics_read(1M)

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
ics_write(1M)

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 writ­
ten 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
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
config(1M)

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(1).

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 ter-
mirated 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 GLO-
BAL. 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> ::= <lto31alphanumericcharor >
  <parametervalue> ::= <ws> <element> { <ws> <element> ... }
<element> ::= <string> | <number> | <substitution> |
  <quote> | <backslash>
<string> ::= <nonsyntaxcharacters>
<number> ::= <decimaldigits> [ "T" ] | <hexdigits> "H"
  | <octaldigits> "Q" | <binarydigits> "Y"
<substitution> ::= "$" <parametername>
Details on MULTIBUS II configuration are available in the MULTIBUS 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.

```plaintext
# # 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(1rref)
5 Device Information

Disk And Tape Device Drivers

5-1
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_LVLAB)
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_ULVLAB)
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_RVLAB)
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_WVLAB)
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_LMDL)
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_ylimdl)
This command returns one entry from the driver’s internal
defect list to the user’s cmdarg structure.

Read Defect List (V_RMDL)
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 cylinder for an ST-506 interface, or on the third
from last cylinder for an ESDI interface.

Write Defect List (V_WMDL)
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

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
console (1M)

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 sys-
            tem. 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 *sys*seg 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:

```c
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:

```c
#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_RALLOC** defines a roving ram disk
- **RAMD_IOC_RFREE** frees a roving ram disk
- **RAMD_IOC_LOAD** loads a ramd partition

The ioctl calls use the ramd_info structure, described above. The ioctl calls are
declared 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.
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:

```c
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_aamin;  /* 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_adow;   /* day of the week, 1-7 */
    unsigned char rtc_mon;    /* month of the year, 1-12 */
    unsigned char rtc_amon;   /* 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 \[ \text{EINVAL} \]
This message is printed if one of the fields used to set the time is outside the prescribed limits.

FILES

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/include/sys/clockcal.h</td>
<td>Driver specific definitions</td>
</tr>
<tr>
<td>/usr/include/sys/rtc.h</td>
<td>Real time clock specific definitions</td>
</tr>
</tbody>
</table>

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.
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_wua**: wakeup address
- **c_devcod**: devices
- **c_level**: interrupt level

These values begin the first line 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 DEVSFLPY (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**

/users/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 car­
   tridge 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(5) man page.

SEE ALSO
   drivers(5), i214(7)
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:

```c
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 iSBC 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/iSBX 351 driver

DESCRIPTION
The i8251 device driver, together with the iasy(7) 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(7) 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(7), console(7)
NAME

ates - asynchronous terminal controller script device driver

DESCRIPTION

The ates 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 ates device driver provides functions which implement the asynchronous terminal protocol.

The ates 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 ates 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 ates line, the ates 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 ates 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/ates - sdevice entries
/etc/conf/pack.d/ates/Driver.o - ates device driver object module
/etc/conf/pack.d/ates/space.c - ates specific declarations and initialization
/usr/include/sys/ates.h - ates specific definitions
/usr/include/sys/atcsmp.h - ates message passing protocol definitions

SEE ALSO

ci(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

```c
  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.

```c
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.

```c
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`.

```c
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:

```c
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.

\[ \text{bpsopen} \]
\[ e\_code = \text{bpsopen}(); \]

The \texttt{bpsopen} function opens the \texttt{bps} device driver. It must be called by all appli-
cations which need access to the \texttt{bps}.

\[ \text{bpsclose} \]
\[ e\_code = \text{bpsclose}(); \]

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

\[ \text{bpsinit} \]
\[ e\_code = \text{bps\_init}(\text{Newbps}); \]
\[ \text{char Newbps}; \]

The \texttt{bps\_init} 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 \texttt{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 \texttt{bps} driver uses this string for references when retrieving
parameters or values. The following functions have the same syntax as in the
Kernel Interface: \texttt{bps\_get\_val}, \texttt{bps\_get\_wcval}, \texttt{bps\_get\_opt},
\texttt{bps\_get\_integer}, \texttt{bps\_get\_range}, and \texttt{bps\_get\_socket}.

**SUPPORTED FEATURES**
The \texttt{bps} driver supports the following \texttt{ioctl()} calls for use by applications:

**BPSINIT**
This command requires that the \texttt{bps} driver be configured with the
\texttt{bps\_testing} flag enabled and that the caller's id is zero. If the argu-
ment \texttt{addr} is NULL, the \texttt{bps} driver reinitializes its internal data struc-
tures with the BPS in RAM. If the argument \texttt{addr} is a pointer to
structure \texttt{bps\_ioctl}, the \texttt{bps} driver reinitializes its internal data struc-
tures with the BPS pointed to by \texttt{bps\_ioctl\_string\_p}. The
\texttt{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 \texttt{bps\_ioctl}. It returns a char-
acter string at the location pointed to by \texttt{bps\_ioctl\_valbuf\_p},
which is the value of the parameter pointed to by \texttt{bps\_ioctl\_string\_p}.

**BPSGETWCPV**
The argument is a pointer to structure \texttt{bps\_ioctl}. It returns a char-
acter string at the location pointed to by \texttt{bps\_ioctl\_valbuf\_p},
which is the value of the wildcard parameter pointed to by
\texttt{bps\_ioctl\_string\_p}.

**BPSGETOPTS**
The argument is a pointer to structure \texttt{bps\_ioctl}. It returns an
integer value at the location pointed to by \texttt{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 argu-
ment is a pointer to structure bps_ioctl. It returns an integer (at
the location pointed to by bps_ioctl->lo_return_p) for the charac-
ter 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:

```c
if !(bpsopen()) {
    BPSstatel = 0;
    while !(bps_get_wcval("ASYNC", &BPSstatel, 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)
            ...}
        }
    }
}
```

10/89
The next example illustrates the format of the BPS for the mapping of host/port to minor numbers shown in the following table.

<table>
<thead>
<tr>
<th>Host id</th>
<th>Port</th>
<th>Minor Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

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:

EFALSE    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
ccl - communications controller interface device driver

DESCRIPTION
The ccl 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 ccl 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 ccl 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 ccl driver to provide management of serial ports on a serial controller board. The ccl 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 ccl device driver.

FILES
/etc/conf/cf.d/mdevice
/etc/conf/sdevice.d/ccl
/etc/conf/pack.d/ccl/Driver.o
/usr/include/sys/ccl.h
/usr/include/sys/ccimp.h

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)

NOTES
DMA to or from iSBX connectors is not yet supported.
NAME
dma – DMA device driver

DESCRIPTION

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)
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 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:

```
ccnet ip /dev/edlina emd 0
```

There should be only one ccnet (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.
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.

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

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(5)ref) 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(5)ref) 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 device drivers(5)ref) 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/ cylinder 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
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
/etc/conf/sdevice.d/i410
/etc/conf/pack.d/i410/Driver.o
/usr/include/sys/ics.h
/etc/conf/pack.d/i410/space.c

mdevice entries
sdevice entries
i410 device driver object module
interconnect space definitions
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 known 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 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.

**MAXCOPYSIZE** 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 better 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

**NOTES**

Specifying `ics_hi_addrics_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 mechan-
                ism 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

ote(liref>
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

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

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_WR_HIWAT (1024 bytes by default) STREAMS blocks messages directed downstream.
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.

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.

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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX PORT</td>
<td>Maximum number of open ports</td>
</tr>
<tr>
<td>MAX_TRAN</td>
<td>Maximum number of outstanding transactions</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>File Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/conf/pack.d/ots/space.c</td>
<td>ots driver configuration file</td>
</tr>
<tr>
<td>/etc/conf/pack.d/ots/Driver.o</td>
<td>ots driver object file</td>
</tr>
<tr>
<td>/etc/conf/node.d/ots</td>
<td>ots driver device node definition file</td>
</tr>
<tr>
<td>/etc/conf/adevice.d/ots</td>
<td>ots driver system device entry</td>
</tr>
<tr>
<td>/usr/include/sys/otsuser.h</td>
<td>ots driver user data structures</td>
</tr>
</tbody>
</table>
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)
Maintenance Commands

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

- dbon
- icsrd, icswr, icsslots, 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 Inter­
rupt 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 systems with a CSM/001.
icsrd(1M)

NAME
icsrd, icswr, icsslots, 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/icsslots
/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 icsslots returns the slot number of the processor it was executed on. By using icsslots, 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 Com-
munications 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(8ire0), reset(8ire0)

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 regis-
ters. Also, care should be taken when using \icswr; with it any board on the bus
can be reset, resulting in loss of data.
NAME
    \texttt{initbp} – initializes the bootstrap parameter string on the processor in the given slot

SYNOPSIS
    \texttt{/usr/lbin/initbp [-v] slot-ID}

DESCRIPTION
    The \texttt{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 \texttt{initbp} command resets the processor in the specified slot after the initialization of the bootstrap parameter string is complete. Command line options are:
    \begin{itemize}
    \item \texttt{-v}  Display diagnostic information.
    \end{itemize}

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 \textit{MULTIBUS II Initialization and Diagnostics Specification}. Also note that any processor can be reset when using \texttt{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

Overview

Architecture
Overview 8-2
Disk Partitions 8-2
Manufacturer's Defect List 8-2
Surrogate Block Pool 8-3
Software Remapping 8-3
Alternate Tracking 8-3
Sector Slipping 8-4
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.
# Appendix A

## Window Interface

<table>
<thead>
<tr>
<th>Windows, What are they?</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using The Mouse</td>
<td>A-1</td>
</tr>
<tr>
<td>Basic Menu</td>
<td>A-1</td>
</tr>
<tr>
<td>Expanded Menu</td>
<td>A-3</td>
</tr>
</tbody>
</table>

| Using The Mouse               | A-3  |
| Basic Menu                    | A-3  |
| Expanded Menu                 | A-7  |
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 relayed) so that all or some of the windows can be viewed at once.
Window Interface

---

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

Menu Selection Area  
(anyplace outside of windows)

Top of MTH Window

---

MSA Bootstrap Loader

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

Console Login:

---

Console Window
SDM Window
Bottom of MTH Window
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.

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

<table>
<thead>
<tr>
<th>Intel System Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pop</td>
</tr>
<tr>
<td>Push</td>
</tr>
<tr>
<td>Pan</td>
</tr>
<tr>
<td>Move</td>
</tr>
<tr>
<td>Resize</td>
</tr>
<tr>
<td>Keyboard Focus</td>
</tr>
<tr>
<td>Pop/Focus</td>
</tr>
<tr>
<td>Pop/Focus/Resize</td>
</tr>
<tr>
<td>Exit</td>
</tr>
</tbody>
</table>

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.
Window Interface

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.

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**

<table>
<thead>
<tr>
<th>Window Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pan</td>
</tr>
<tr>
<td>Attach Keyboard</td>
</tr>
<tr>
<td>Pop to Foreground</td>
</tr>
<tr>
<td>Push to Background</td>
</tr>
<tr>
<td>Pop and Set Focus</td>
</tr>
<tr>
<td>Move Window</td>
</tr>
<tr>
<td>Resize Window</td>
</tr>
<tr>
<td>Expand Window</td>
</tr>
<tr>
<td>Reduce Window</td>
</tr>
<tr>
<td>Map Window</td>
</tr>
</tbody>
</table>
Window Interface

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.
Window Interface

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.
See the iSBX 279 Display Subsystem Installation Guide.
Appendix B

Information Unique to MULTIBUS Systems
Volume Label: ivlab Structure  B-1
Manufacturer's Defect List: MDL Structure  B-3
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 */
};
```
Information Unique to MULTIBUS Systems

\( v_{\text{name}} \)  This is the name of the volume ASCII, right blank-filled.

\( v_{\text{flags}} \)  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:

\[
\begin{align*}
\#define \text{VF\_AUTO} & \ 0x01/* 1\rightarrow\text{byte is valid */} \\
\#define \text{VF\_DENSITY} & \ 0x02/* 0=FM, 1=MFM */ \\
\#define \text{VF\_SIDES} & \ 0x04/* 1=double-sided */ \\
\#define \text{VF\_MINI} & \ 0x08/* 0=8, 1=5.25 */ \\
\#define \text{VF\_NOT\_FLOPPY} & \ 0x10/* 0=flop track 0 is 128SD 1=not floppy */
\end{align*}
\]

\( v_{\text{driver}} \)  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.

\[
\#define \text{UNIX\_FD} \ 6/* UNIX file-driver number */
\]

\( v_{\text{gran}} \)  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.

\( v_{\text{size\_l}}, v_{\text{size\_h}} \)  Low and high order bytes of volume size expressed in bytes. Not used by System V/386.

\( v_{\text{maxfnode}} \)  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.

\( v_{\text{stfnodes\_l}}, v_{\text{stfnodes\_h}} \)  Low and high order bytes of start of fnodes in RMX. Set to 2, not used by System V/386.

\( v_{\text{szfnode}} \)  Size of an fnode, in bytes. Set to size of an inode: 32.

\( v_{\text{stfnode}} \)  Start of fnodes. Set to root inode: 2.

\( v_{\text{devgran}} \)  Device granularity, in bytes per sector. Physical device granularity, determined when device was formatted.
Information Unique to MULTIBUS Systems

\( v_{\text{intl}} \)  
Physical device interleave. When set to 0, indicates unknown. Not used in System V/386.

\( v_{\text{trskew}} \)  
Track skew. When set to 0, indicates unknown. Not used in System V/386.

\( v_{\text{sysid}} \)  
ID of operating system that formatted volume. Set to 0x0040 for System V/386.

\( v_{\text{sysname}} \)  
Name of operating system that formatted volume. Set to "System V:" for System V/386. Twelve ASCII characters, right blank-filled.

\( v_{\text{dspecial}} \)  
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.

\( \text{dr}_{\text{nalt}} \)  
This field changes to a density flag if the volume label is on a floppy device. \( \text{dr}_{\text{nalt}} \) set to 0 indicates a single-density, FM-format floppy, and \( \text{dr}_{\text{nalt}} \) set to 1 indicates a double-density, MFM-format floppy.

\( v_{\text{fsdelta}} \)  
The absolute physical sector number of the start of the root file system.

\( v_{\text{freespace}} \)  
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).

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

struct st506hdr {
  unsigned short bb_valid;
  unsigned short bb_num;
};

struct st506defect {
  unsigned short be_cyl;
  unsigned char be_surface;
  unsigned char be_reserved;
};
```
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
MULTIBUS II Interconnect Interface Specification
MULTIBUS II Message Passing Coprocessor External Specification
MULTIBUS II MPC User’s Guide