CONVEX Multibus SMD Disk (*dev4100*) Diagnostics Manual

Document No. 760-001930-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus Storage Module Drive (SMD)
Disk (dev4100) Diagnostics Manual
Order No. DHW-231
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus Storage Module Drive (SMD) Disk (dev4100) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-001930-000	May 1991	First release. Contains the *dev4100* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

A Reporting Problems

List of Tables

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4100* diagnostic, which checks the Xylogics 450/451 controller and any attached Storage Module Drive (SMD) devices. This document is not a tutorial, but rather a reference for the users of the *dev4100* diagnostics, including field service and manufacturing test personnel, as well as the diagnostics sustaining staff. In addition, CONVEX customers can use this manual to execute the *dev4100* diagnostic.

Scope

This manual applies to all CONVEX computers.

Organization

This document consists of the following:

- **Chapter 1. Diagnostics Environment**—Introduces the theories and concepts that underlie I/O diagnostics on CONVEX machines as well as the basic overview, philosophy, and structure of I/O diagnostics.
- **Chapter 2. EGOS Overview**—Provides a brief overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing.
- **Chapter 3. Dshell Overview**—Provides a brief overview of and a general introduction to the *dshell* utility.
- **Chapter 4. Multibus SMD Disk Test (*dev4100*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes the disk parameters file, the diagnostic cylinder description, and controller and drive error codes and messages.
- **Appendix A. Reporting Problems**—Provides an example of the CONVEX *contact* utility for reporting minor software and hardware problems.

Notational Conventions

The notational conventions used in this text are listed below:

- Bit numbering is left to right, `N-1` through `0`. The most significant numerical bit is `N-1`, the least significant `0`. The bit numbering represents the binary weight of a position.
- Bit fields are specified using the following convention: *name*<*x..y*> where the bit field is *name* from bits *x* through *y*.
- Individual bit positions within a register are denoted by specific positions separated by commas. For example, REG<15,4,0> denotes bits 15, 4, and 0 of REG.
- Byte numbering is from left to right
- A *bit* is a single binary value or entity
- A *nibble* is 4 bits
- A *byte* is 8 bits
- A *halfword* is 16 bits
- A *word* is 32 bits
- A *longword* is 64 bits
- *Single precision* is a 32-bit floating point word
- *Double precision* is a 64-bit floating point longword
- An *instruction* is a multihalfword operand
- A bit is *set* when it contains a binary value of `1`.
- A bit is *clear* when it contains a binary value of `0`.
- All memory and I/O addresses are written in hexadecimal notation unless explicitly stated otherwise.
- All register contents are written in hexadecimal notation unless explicitly stated otherwise.
- A *register* is a programmer-visible hardware storage element internal to the processor
- *Physical memory* is the physical storage installed in the processor
- *Virtual memory* is the perceived amount of physical memory as seen by the application programmer
- The symbol *K* is an abbreviation for *kilo* or 1,024
- The symbol *M* is an abbreviation for *mega* or 1,048,576
- The symbol *G* is an abbreviation for *giga* or 1,073,741,824
- A *stack* is a linked-list group of words useful for dynamic allocation and deallocation of memory
- A *return block* is a collection of registers that is pushed or popped from a context stack in response to an instruction or other event
- *Reserved* or *undefined* convey what to expect, if anything, from unused fields in registers, reserved memory, or reserved I/O space. Algorithm implementation based on the use of undefined or reserved fields is not recommended.

Warnings

The following are examples of warnings, cautions, and notes and their typical content as used in CONVEX documents:

WARNING

Warnings highlight procedures or information necessary to avoid injury to personnel. A warning immediately precedes the critical information and includes a description of the hazard.

CAUTION

Cautions highlight procedures or information necessary to avoid damage to equipment, loss of data, or invalid test results. A caution immediately precedes the critical information and includes a description of the possible damage.

NOTE

Notes highlight useful information that is supplemental in nature. A note may immediately precede or follow the information that is being highlighted.

Associated Documents

The following is a partial list of other manuals or books that may provide more detailed information on the topics presented in this manual:

- *CONVEX Processor Diagnostics Manual (C1, C120)*, Order No. DHW-071
- *CONVEX Processor Diagnostics Manual (C200 Series)*, Order No. DHW-081
- *CONVEX Architecture Reference*, Order No. DHW-005
- *CONVEX SPU UNIX Utilities Manual*, Order No. DHW-021
- *CONVEX Processor Operation Guide (C100 Series, C200 Series)*, Order No. DHW-015
- *CONVEX Diagnostic Utilities Manual (C1, C120)*, Order No. DHW-072
- *CONVEX Diagnostic Utilities Manual (C200 Series)*, Order No. DHW-082
- *CONVEX UNIX Tutorial Papers*, Order No. DSW-002
- *The C Programming Language*, Kernighan & Ritchie, Order No. DSW-046

Ordering Documentation

To order the most current version of this or any other CONVEX document, use the product number. If the product number is not known, order by the exact title. In some situations, the most current version may not be desired. To receive a specific version of a manual, order the manual by its document number, or part number, which can be obtained by contacting the local CONVEX office or by calling the Technical Assistance Center.

The product number for this manual is DHW-231.
The document number for this manual is 760-001930-000.

CONVEX documents can be ordered by mail by sending a request to:

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

Technical Assistance

Hardware and software support can be obtained through the CONVEX Technical Assistance Center (TAC):

- From all locations in the continental United States, call 1(800)952-0379.
- From locations in Alaska, Hawaii, and Canada, call 1(214)497-4379.
- From all other locations, contact the nearest CONVEX office.

Reader's Forum

If you wish to mail your comments to us, please use the form at the end of this manual and list the document page number with your questions and comments. Thank you.

Chapter 1
Diagnostics Environment

1.1 Overview

CONVEX system diagnostics consist of a suite of test programs designed (except where noted) to execute under the Service Processor operating system, SPU UNIX. These programs utilize the capabilities of the Service Processor to test the operation of one or more of the functions of the system and report any errors detected. All of the diagnostics in this manual are intended to be executed "off-line"; that is, while CONVEX UNIX is not being executed by any of the Central Processing Units (CPUs) in the system.

The Service Processor, together with SPU UNIX, various diagnostic utilities, and the test programs, themselves, comprise the CONVEX diagnostic environment. This chapter describes the hardware and software components of this environment and is intended to provide the background necessary to fully utilize the capabilities of the CONVEX processor diagnostics.

For more information about the diagnostic environment refer to the Diagnostic Environment chapter in the *CONVEX Processor Diagnostics Manual (C200 Series)* or the *CONVEX Processor Diagnostics Manual (C1, C120)* depending on the architecture of the machine under test.

1.2 Test Program Naming Conventions

Test program names are in the form *cattypedevnn.suffix* where:

- *cat* is the subsystem being tested
- *type* is the type of test being performed, e.g., standalone, self-test, or offline functional test
- *dev* is the device being tested, e.g., disk, tape, or printer. This segment of the test program name is used *only* if the category is a device.
- *nn* is a CONVEX code used for distinguishing between test programs
- *suffix* is one of three program identifiers:
  - *.t* are programs that execute on SP2
  - *.x00* and *.rnn* are object files for different target processors other than the SP2. The target processor depends on the subject of the test. The test program name must have the test program category (*cat*) at the beginning of the name to determine the target processor.

1.2.1 Test Program Categories

Test program categories include those tests for the CPU, peripheral devices, I/O system, memory system, SP2, and entire system. For example, *cpu4041* is a CPU vector instruction test while *mem4000* is a memory system functional test. The following table lists test program categories:

Table 1–1, Test Program Categories

TEST PROGRAM CATEGORIES	
Test Category (*cat*)	**Description**
cpu	CPU subsystem related test
dev	Peripheral device test
io, idc, tli	I/O subsystem related test
mem	Memory subsystem related test
spu	SP2 subsystem related test

1.2.2 Test Program Types

A test program type describes whether a test is a standalone test, self-test, kernel hardware test, or an offline or online functional test. See the following table for the numbering system and description of test program types:

Table 1–2, Test Program Types

TEST PROGRAM TYPES	
Number (*type*)	**Description**
0	Standalone test
1	Self-test
2	Kernel hardware test
4, 5	Offline functional test

1.2.3 Test Program Device Types

Test programs will test disks, tapes, terminals, printers, and networks. See the following table for the numbering scheme and a description of the test program device types:

Table 1–3, Test Program Device Types

TEST PROGRAM DEVICE TYPES	
Number (*dev*)	**Description**
1	Disk
2	Tape
3	Terminal
4	Printer
5	Network

1.2.4 Examples of Test Program Names

The following table presents some examples using the naming conventions outlined above:

> **NOTE**
>
> In the following table, SOFF stands for Standard Object File Format.

Table 1–4, Example Test Program Names

EXAMPLE TEST PROGRAM NAMES	
Test Program Name	**Description**
cpu4041.t	SP2 object code in *b.out* format for *cpu4041*
cpu4041.rnn	C210 or C220 machine object code in SOFF format (relocatable)
cpu4041.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
mem4000.t	SP2 object code in *b.out* format for *mem4000*
mem4000.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
dev4100.t	SP2 object code in *b.out* format for *dev4100*
dev4100.x00	IOP object code in *b.out* format

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2
EGOS Overview

2.1 Overview

This chapter provides an overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing. There are three basic types of EGOS systems, one for each type of CCU. There is one for the Multibus interface, one for the VME interface, and one for the HIA interface. This chapter will explain the three types of EGOS systems and how EGOS is positioned within the overall operating system environment.

2.2 Purpose of EGOS for Diagnostic Testing

EGOS is basically a simple operating system that the device tests use to handle interrupts, schedule processes, and generally allocate and control IOP/VIOP resources. The diagnostics code uses both EGOS and the Message Based System (MBS) to manipulate test program control over to the CCU side of the test program. MBS is not a part of EGOS but rather a system that allows a common section of memory to be used as a message area between multiple processors. For more information on MBS, refer to the *CONVEX Guide to Writing Device Drivers.*

EGOS initially sets up interrupt tables, determines how many chassis there are, and initializes its windows and resource allocation tables.

2.3 EGOS for the Multibus Interface

EGOS for the Multibus interface supports event driven device drivers. The Multibus version of EGOS takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.4 EGOS for HSP Interface, HSP EGOS

EGOS for the HSP interface supports event driven device drivers. The HSP version of EGOS is like the Multibus version. It takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.5 EGOS for VME Interface, VIOP EGOS

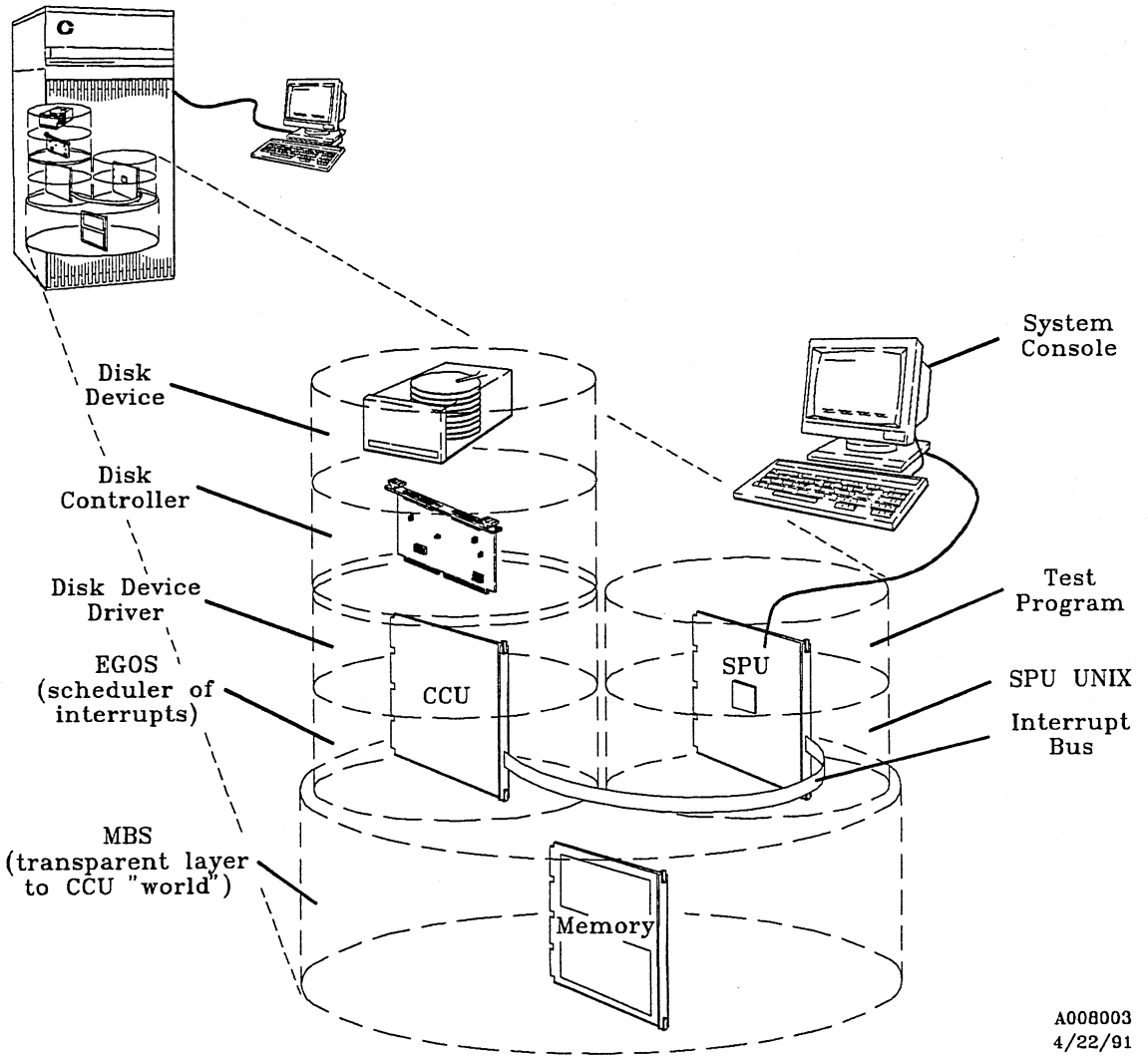
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes it less complicated to create user written device drivers. Each process device driver has a priority level that can be defined relative to other processes. The scheduler supports 32 process priorities and is preemptive for higher priority processes. The VIOP hardware supports 14 device events for event driven device drivers. The 14 levels actually share 2 68020 interrupt levels. Therefore, two is the maximum number of processes at any given time.

2.6 EGOS Position in the Environment

EGOS is positioned in the operating environment between the actual device driver and MBS. MBS is a transparent layer that bridges the CCU and its resources to SPU UNIX. SPU UNIX handles many of the message manipulations that occur during testing. Many error messages that occur during diagnostics testing come from the device driver. When the device driver detects an error from the controller, it calls a routine in EGOS that places a message in the MBS system. This causes SPU UNIX to be interrupted and it retrieves the message from MBS. SPU UNIX then passes a signal to the test program. The test program then prints an error message to the console based on the code that it received.

The following figure illustrates the position of EGOS in the operating system environment.

Figure 2–1, EGOS' Position in the Environment



THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3
Dshell Overview

3.1 Overview

This chapter provides a brief overview of the *dshell* utility. Included in this overview is an overall explanation of the utility and a list of the utility's commands. For a complete description of this utility, refer to the Dshell chapter of the *CONVEX Diagnostic Utilities Manual (C200 Series)* or the *CONVEX Diagnostic Utilities Manual (C1, C120)* depending on the architecture of the machine under test.

3.2 Diagnostic Shell (*dshell*) Overview

The Diagnostic Shell (*dshell*) is a command interface program that runs on the Service Processor. Most of the diagnostics available for the CONVEX machines are interfaced through the *dshell*. Certain peripheral diagnostics are run as standalone tests. To determine whether a test can be run under the *dshell*, consult the appropriate chapter in this manual.

The *dshell* has two basic functions:

- Selecting diagnostics for execution
- Selecting test options
  - Pause on a failure or at the beginning or end of any specific subtest
  - Loop on a specific type of subtest or on a given set of subtests
  - Select subtest execution order
  - Direct test output to a file or to the screen (or both) to monitor the test as it runs or to analyze test results later
  - Select long or short error messages, or turn messages off
  - Execute either user-created or predefined command scripts

The following table list the various *dshell* commands and their functions.

Table 3–1, *dshell* Commands

COMMAND	FUNCTION
! [command]	This command is used to access, or *fork* a UNIX shell to execute the command that follows *!*.
exit	The *exit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
quit	The *quit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
^C	Returns user to the *dshell* command level if no subtest is running.
^B	Immediately terminate the *dshell* and any associated active processes. Core is dumped.
help	The *help* command causes a standard *help* menu to be displayed. The menu describes the correct command syntax for each *dshell* command and gives a terse description of what each command does.
status	The *status* command generates a report on the current state of the *dshell* command options. This report gives the name of each flag, its current value, and an explanation of its current effect.
log [options]	The *log* command provides a mechanism for specifying the number of failures that will be allowed to occur before a test or subtest terminates execution.
loop [options]	The *loop* command causes the *dshell* to repeat the execution of a test or subtest.
msgs [options]	The *msgs* command enables or disables different levels of test, class, and subtest result messages.
pause [options]	The *pause* command returns program control to the *dshell* to the beginning, end, or failure of all or specific subtests.
test [options]	The *test* executes specific tests, and displays test, class, and subtest menus.

3.3 Syntax Help for *dshell* Commands

The syntax for each *dshell* command can be obtained by typing the command with no options and pressing <**CR**>. For example, by entering **loop** and pressing <**CR**>, the syntax help in the following figure will be displayed on the screen:

Figure 3–1, Syntax Help for the *loop* Command

```
: loop
Proper syntax is:

loop off (-s) (-t)                          :disables loop modes
loop -s nnn                                 :loop on subtest nnn
loop -t                                     :loop on test
```

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4
Multibus SMD Disk Test (*dev4100*)

4.1 Overview

The *dev4100* test checks the Xylogics 450/451 controller and any attached Storage Module Drive (SMD) devices.

4.2 Prerequisites and Required Equipment

This test is dependent on the existence and correctness of the SPU disk file */mnt/bin/lib/DB_diskfmt*. Refer to the section "Disk Parameters File, *DB_diskfmt* Description" in this test description for more information about this file.

The test requires the system configuration in one of the next two tables depending on the type of machine under test.

Table 4–1, Hardware Requirements (C1, C120)

ITEM	MINIMUM	MAXIMUM
Xylogics 450/451 controller	1	12
SMD device	1	12
Input/output processor (IOP)	1	5
Multibus card cage	1	10 (2/IOP)
Service processor unit (SPU)	1	1
Multibus control unit (MBCU)	1	12
Memory control unit (MCU)	1	1
Memory array unit (MAU)	4MB	System limit

Table 4–2, Hardware Requirements (C200 Series)

ITEM	MINIMUM	MAXIMUM
Xylogics 450/451 controller	1	12
SMD device	1	12
Input/output processor (IOP)	1	12
Multibus card cage	1	12
Service processor unit (SP2)	1	1
Multibus control unit (MBCU)	1	12
Memory system (one odd, one even)	1	System Limit
Peripheral interface adapter (PIA)	1	4
CPU utilities (CPX)	1	1

If a SMD is not available, only some of the Class 1 subtests can be run to check a Xylogics 450/451 controller. (Refer to Table 4-5, Class 1 Subtests in this chapter for the Class 1 subtests that can be executed without a SMD device.)

4.3 Test Invocation

The *dev4100* test executes under the Diagnostic Shell (*dshell*) and supports all the features of the *dshell*. The *dshell* permits tests to be initiated in any order.

To invoke the *dev4100* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses appear sequentially on the screen, one line at a time. All prompts and responses are shown in one figure for convenience.

Figure 4–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4100 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

NOTE

After entering **dshell**, specific *dshell* parameters may be changed. Refer to the "Dshell Overview" chapter of this manual for more information.

Entering only **test dev4100** executes all *dev4100* subtests sequentially. Execute a specific class(es) of subtest(s) or one or more individual subtests by using the *-c* or *-s* options, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter of this manual. The [+>*filename*] option allows the test results to be appended to *filename*.

The following alternate test invocation procedure may be required in some cases.

CAUTION

The user response, **initall**, is typically required if the *initall* utility has not been run since the last power up. However, if any problems have occurred subsequent to the last time *initall* was run, (i.e., system crash, hard error, or failure of previous diagnostic), it should be run again. In this case, failure to run *initall* could result in invalid test results.

NOTE

The *initall* utility requires a significant amount of time (2 to 3 minutes depending on whether the control stores have been previously loaded) to execute. If no system abnormalities have occurred subsequent to the last time the system was booted or *initall* was executed, it is not necessary to run *initall.*

Figure 4–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4100 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

4.3.1 Test Parameter Menu

Once the test is invoked, a test menu prompt is presented allowing selection of default switches. The following figure shows all prompts, their possible answers (in brackets [ ]), and their default answers (in parentheses ( )). The prompts and responses in the following figure appear sequentially on the screen, one line at a time. All the prompts and responses are shown in one figure for convenience.

To receive help or information during test parameter entry, enter one of the following characters followed by RETURN:

Table 4–3, Getting Help During Test Parameter Entry

CHARACTER	DESCRIPTION
?	Provides the help information
d	Displays *diskfmt* file containing the drive parameters
e	Displays drive entries you have made
h	Provides help for a specific prompt
i	Displays the */ioconfig* file

After the desired help information is displayed, the system beeps and redisplays the last prompt.

The **Test Parameter Menu** figure illustrates *all* prompts that can be asked during test parameter input. However, some prompts may be omitted, depending on answers to previous questions. In all cases, questions are numbered sequentially. However, the numbers displayed during testing may not correspond to those shown in the figure, as the questions illustrated are examples only.

Figure 4–3, Test Parameter Menu

```
                    ENTER TEST PARAMETERS
       []     Encloses allowed input ranges or values
       ()     Encloses the default value
       ^      Returns to the previous prompt
       :nn    Returns to the prompt # nn
       :      Returns to the first unsatisfied prompt
       :?     Reviews previous entries
?      Prints an additional help menu

 1:  Are writes to disks allowed [y,n]              (n) ->
 2:  Number of errors allowed per device each subtest
    [0-65535]                                       (0) ->
 3:  Number of devices failed after which test aborts
    [0-65535]                                       (0) ->
 4:  Run subtest 112 to check Attn/Ack [y,n]        (n) ->
 5:  Ioconfig file []                        (/ioconfig) ->

              PERIPHERAL CONFIGURATION DATA[1]
      CCU    Chassis  Type      CSR    Int  Unit  TYPE
     -------- -------  --------  -----  ---  ----  --------
 1)  iop  4     0   DKC-001   0x3f0    2     0    DKD-008
 2)  iop  4     0   DKC-001   0x3f0    2     1    DKD-005
 3)  iop  4     0   DKC-001   0x3f0    2     1    DKD-005
 4)  iop  4     0   DKC-001   0x3f0    2     0    DKD-008
 5)  iop  4     0   DKC-001   0x3f0    2     1    DKD-008

Enter device 99 to begin user-defined configurations or 0 to end selection
 6:  Device selection [0,1-5,99]                   (0) ->
 7:  Already formatted [y,n]                       (y) ->
Enter device 99 to begin user-defined configurations or 0 to end selection

 8: Device selection [0,1-5,99]                    (0) ->

Enter IOP -1 to end user-defined configurations

 9:  IOP [0,3-7]                                    () ->
10:  Multibus Chassis [0-3]                         (0) ->
11:  Controller Offset in Multibus [0x0-0xfff]
                                                 (0x3f0) ->
12:  Interrupt number [0-7]                         (2) ->
13:  Unit number [0-3]                              (0) ->
14:  Drive name []                           (DKD-005) ->[2]
15:  Already formatted [y,n]                        (y) ->

Enter IOP -1 to end user-defined configurations

16:  IOP [0,3-7]                                    () ->
17:  Enter OK, or :NN to return to question NN [OK]
                                                   (OK) ->

          ********** WARNING!  ***********
        THIS TEST IS POTENTIALLY DATA DESTRUCTIVE!
        IF YOU CONTINUE, DATA COULD BE LOST!

Do you want to continue [yn]      ->
```

[1] The configuration data is only displayed once; to display it again, type **i** (RETURN) at any prompt during test parameter query.

[2] The drive name must match with a drive in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*). To display the *DB_diskfmt* file, type **d** (RETURN) at any prompt.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parameter Menu. The following list summarizes the available options:

:nn — Returns to an earlier prompt (n is the prompt number)

: — Advances to the next unanswered prompt

:? — Displays (reviews) all responses up to the current prompt

? — Requests help for the current prompt (if available)

^ — Returns to the previous prompt

4.3.2 Prompt Explanations

The test parameter prompts are repeated and explained in the following paragraphs:

```
Are writes to disks allowed [y,n]                    (n) ->
```

Enter **y** to allow writes to the selected disks. The test itself screens all commands before they are allowed to execute and inhibits any operations which are data destructive regardless of whether the write protect switch on the drive is set or not.

```
Number of errors allowed per device each subtest
[0-65535]                                            (0) ->
```

Enter the number of errors that can occur and still allow a device to continue. For instance, if **3** is entered, a device can have 0-3 errors and continue running. On the fourth error, the device stops and is not restarted for the duration of the test.

```
Number of devices failed after which test aborts
[0-65535]                                            (0) ->
```

Each time a unit fails (exceeds the subtest error limit), a count of the number of failed devices is incremented. When a controller fails, the count is incremented for the controller and for each active drive on the controller. If this count exceeds the number of devices specified at this prompt, the entire *dev4100* test aborts.

```
Run subtest 112 to check Attn/Ack [y,n]              (n) ->
```

> **NOTE**
>
> This feature is not used in the OS driver, so this feature does not affect the use of the controller; therefore, enter **n**. Refer to Subtest 112, Verify Attention Request/Acknowledge for more information.

Enter **y** to execute Subtest 112, Verify Attention Request/Acknowledge. Subtest 112 verifies an active controller will stop processing and acknowledge an attention request which allows the CCU driver to add or remove Input/Output Parameter Blocks (IOPBs) from an active chain. Enter **n** and Subtest 112 will not be executed.

```
Ioconfig file []                                    (/ioconfig) ->
```

This test query allows selection of a file that contains peripheral device descriptions. The default displays the system's configuration file. To display an I/O configuration file previously created, enter the filename. The configuration file is then displayed. (A relative or absolute path may be specified.)

If drives are to be tested with other than the expected configuration, a new I/O configuration file may be created. As an alternative, nonstandard configurations may be entered by responding to the user-defined configuration test parameters (questions 9-16 as shown in Figure dev4100-3, Test Parameter Menu). Then the drive and/or controller does not have to be defined in an I/O configuration file. Drives from an I/O configuration file can be selected and also drives can be defined with user-defined configurations.

```
Device selection [0, 1-5, 99]                              (0) ->
```

Enter the number of the device to be tested. Only one device can be selected with each device selection prompt. A total of 12 devices can be selected in any order.

If **99** is entered, the following four test parameters do not display. Instead, the user-defined prompts are displayed, beginning with prompt `IOP [0, 3-7]`. If **0** is entered, user selection of devices terminates and the prompt `Enter OK` is displayed.

```
Already formatted [y,n]                                    (y) ->
```

If **y** is entered the drive must be formatted prior to running *dev4100* Class 1 subtests.

```
Device selection [0, 1-5, 99]                              (0) ->
```

Enter the number of another device from the I/O configuration file. If **0** is entered, selection of devices is terminated. If **99** is entered, prompts are displayed that allow configurations to be defined that do not exist in the I/O configuration file.

```
IOP [0, 3-7]                                               () ->
```

This prompt begins the queries for user-defined drives; additional prompts are displayed requesting hardware configuration information. Enter the CCU slot number for the desired IOP or **-1** to terminate device selection. On a C1 or C120, the valid range of IOP numbers is 3-7. On a C200 Series machine, the valid range for IOP numbers is 0-3.

```
Multibus chassis [0-3]                                     (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                                       (0x3f0) ->
```

Enter the low-order 12 bits of the controller's address within the Multibus.

```
Interrupt number [0-7]                                    (2) ->
```

Enter the interrupt level of the controller within the Multibus.

```
Unit number [0-3]                                         (0) ->
```

Enter the unit number of the drive to be tested. The unit number entered should also be the unit selected on the drive. (Up to four drives can be attached to one controller; each drive is assigned a unit number which is usually switch selectable on the drive.)

```
Drive name []                                        (DKD-005) ->
```

Enter the drive name. The response to this query must be in the form of DKD-xxx and the name must match one of the drive definitions in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*).

```
Already formatted [y,n]                                   (y) ->
```

Enter **y** if the drive has been previously formatted by the utility *diskfmt*; otherwise, enter **n**. If **y** is entered, time is saved because the subtests do not need to format the disk before starting read and write I/O operations.

```
IOP [0,3-7]                                               (0) ->
```

Enter the CCU slot number for the desired IOP to set up test parameters for another user-defined drive; otherwise, enter a **-1** or RETURN to terminate device selection. On a C1 or C120, the valid range for IOP numbers is 3-7. On a C200 Series machine, the valid range for IOP numbers is 0-3.

```
Enter OK, or :NN to return  to question NN [OK]
                                                      (OK) ->
```

If **OK** or RETURN is entered, the test parameter menu terminates and all inputs are no longer changeable.

```
Do you want to continue [yn]                                ->
```

Enter RETURN to begin test execution, or enter **n** to abort the *dev4100* test. When all prompts have been answered, the screen displays a test parameter summary which echos the prompts that have been answered. The following figure illustrates an example of a "Test Parameter Summary" screen. The actual values and responses vary according to the input.

Figure 4–4, Sample Test Parameter Summary

```
                TEST PARAMETER SUMMARY

Are writes to disks allowed                                  :
Number of errors allowed per device each subtest             : 0
Number of devices failed after which test aborts             : 0
Run subtest 112 to check Attn/Ack                            :
Ioconfig file                                                : /ioconfig
Enter OK, or :NN to return to question NN                    : OK

                        DRIVE CONFIGURATION DATA

        IOP Mbus Mbus  Int   Unit  #    #  Phys  Log Pre-
         #   #   CSR  Level   #   Cyl  Hds  Sec  Sec Fmtd   Name
        --- --- ----- -----  ---  ---  ---  ---  ---  ---  -------
   1.    4   0  0x3f0   2     0  1024   27   68   67   y    DKD-008
1000.    4   0  0x3f8   3     1   760   19   60   59   y    DKD-005

Performing sysreset of ccus and memory...
Initializing I/O subsystem and Loading IOP(s)
```

4.4 Hardware Initialization Sequence

After the last prompt is entered, and before test code execution, the following events occur:

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The IOP is booted and loaded
- A driver on the IOP is started
- IOP local test variables are initialized

After all the above events have occurred, the test code is started.

4.5 Class Descriptions

This test contains the following two classes of subtests which are listed in the following table:

Table 4–4, *dev4100* Test Classes

CLASS	DESCRIPTION
1	Controller functionality tests
2	Drive functionality tests

4.5.1 Class 1 Subtests

Class 1 subtests exercise all the commands of the Xylogics 450/451 controller to verify its functionality. The subtests contained in *dev4100* are shown in the following table:

Table 4–5, Class 1 Subtests

SUBTEST	DESCRIPTION	SMD REQUIRED	WRITE REQUIRED	TIME (min:sec)
100	Performs controller reset and checks default drive status for each drive			0:01
101	Executes self-test command			0:01
102	Executes DMA test command			0:01
103	Performs controller buffer write/read commands (max throttle)			0:01
104	Verifies command chaining			0:01
105	Performs NOP command; verifies that specified drives are online	x		0:01
106	Executes set drive size and read drive status commands			0:01
107	Formats a track of diagnostic cylinder and reads track header; uses all interleaves	x	x	0:02
108	Verifies write track headers command; reads back with read track headers command	x	x	0:01
109	Verifies write and read (with verify) commands (all throttle settings)	x	x	0:02
110	Verifies seek command	x		0:01
111	Performs write and read header, data, and ECC command	x	x	0:01
112	Verifies attention request/acknowledge	x		0:02
113	Verifies that controller rejects bad heads, cylinders, sectors, and a bad command	x	x	0:01

4.5.1.1 Subtest 100, Perform Controller Reset and Read Drive Status Command

Subtest 100 performs a controller reset on each controller and verifies that the reset works by checking the address and relocation registers of the controller. It also performs a pattern test of these registers with walking '0's and '1's patterns.

4.5.1.2 Subtest 101, Execute Self-test Command

Subtest 101 executes each controller's self-test by first using polling, then using interrupts in place of polling, and repeating the self-test.

4.5.1.3 Subtest 102, Execute DMA Test Command

Subtest 102 executes a Direct Memory Access (DMA) test, transferring patterns that check every data line to each controller. Each pattern consists of one data bit turned on, while other bits are

set to zero. The test causes the *on* bit to rotate through each data bit to verify that each bit can assume an *on* and *off* state and to check that no data bit affects other data bits.

4.5.1.4 Subtest 103, Perform Controller Write/Read

Subtest 103 performs a 512 byte write to and read from the memory buffer in each controller to verify that the *load* and *dump* buffer commands work. During testing, the throttle setting is set to 128 words per DMA burst, which is the maximum allowed. The subtest writes and verifies a `0x55` pattern followed by a second write and verify using a `0xAA` pattern.

4.5.1.5 Subtest 104, Verify Command Chaining

Subtest 104 verifies that command chaining works by chaining a write followed by a read of a 512 byte buffer within the controller. If other controllers are available, then this subtest executes until all controllers are checked for chaining. The test performs two write/read sequences per controller. The first write uses a repeating pattern of `0x55`; the second, a `0xAA` pattern.

4.5.1.6 Subtest 105, Perform NOP Command

Subtest 105 performs a *NOP* command on all drives selected via the test parameter queries. The *NOP* command selects a drive, checks for drive ready, and then de-selects the drive. If any of the selected drives show up as not ready, this test fails.

4.5.1.7 Subtest 106, Execute Set Drive Size and Read Drive Status Commands

Subtest 106 performs a *read drive status* command on all drives of every controller. It uses the *set drive size* and *read drive status* commands to verify that the configuration can be changed.

4.5.1.8 Subtest 107, Verify Format and Read Track Headers Commands

Subtest 107 formats track 0 on the diagnostic cylinder and read verifies the success of the format by performing a *read track headers* command. The test checks this information to verify an interleave of 1 and the correct number of headers. Then the test attempts interleaves of 2, 3, 4, etc., through an interleave of 16 and verifies the interleave was formatted correctly.

4.5.1.9 Subtest 108, Verify Write Track Headers Command

Subtest 108 verifies the *write track headers* command by performing a *read track headers* on track 1 of the diagnostic cylinder. It then writes the headers back out with all head sector numbers set to 0. The test verifies the changed headers by performing another *read track headers.*

4.5.1.10 Subtest 109, Verify Write and Read Commands

Subtest 109 ensures that the *write* and *read* commands work. This subtest first identifies two consecutive sectors on track 2 of the diagnostic cylinder. Then it writes and read verifies the following data:

Table 4–6, Subtest 109, Write and Read Data

SECTOR (of the two chosen)	PATTERN	SECTOR
1st	0x55	Single sector I/O
1st	0xAA	Single sector I/O
Both	0x55	Multisector I/O
Both	0xAA	Multisector I/O

4.5.1.11 Subtest 110, Verify Seek Command

Subtest 110 verifies the *seek* command completes without error. First it seeks to the diagnostic cylinder and confirms its position by reading track headers. A *seek* is then executed to the maximum cylinder and a *read track headers* is again used to confirm positioning.

> **NOTE**
>
> Although the *seek* command completes without error, this subtest may not detect that the controller did not seek the drive. Although a *read track headers* command is issued after the seek to verify the head is on the correct cylinder, the head may get to that cylinder through the implied seek of the *read track headers* command instead of by the *seek* command. Thus, an error in the *seek* command would not be detected. This also implies that the controller did not update its internal cylinder position when the *seek* command was issued, because, if it had, the implied seek would not occur. For these reasons, it is highly unlikely that this test would fail to detect a seek error.

4.5.1.12 Subtest 111, Perform Write and Read Header, Data, and ECC

Subtest 111 reads header, data, and Error Correcting Code (ECC) information from each drive and writes it back. Header, data and ECC are known as HDE. Before beginning, it locates two consecutive sectors on track 3 of the diagnostic cylinder. The test writes and read verifies these sectors with test patterns 0x55 and 0xAA, respectively. These sectors are read using the *read HDE* command. It verifies the headers and data fields, and then swaps data and data ECCs between the two sectors. Next, the *write HDE* command writes the sectors back out, and then the test issues the *read HDE* command to read the data back from the two sectors. When the test completes, the data should be in the opposite sectors from where it was originally written.

4.5.1.13 Subtest 112, Verify Attention Request/Acknowledge

Subtest 112 verifies that the attention request/acknowledge feature of the controller works. This technique is used when the controller driver wants to add parameter blocks onto an already executing chain.

> **NOTE**
>
> This feature does not work at the time of this printing because the controller has a known flaw; therefore, Subtest 112 should not be enabled. It is also not used in the OS driver. If this feature is corrected in the future, this test should be enabled.

The test builds a chain of four *read track headers* commands to each drive on the minimum, maximum, and back to the minimum cylinder to ensure they do not complete before the last two parameter blocks are added to the chain. The last parameter block performs one more minimum-maximum track seek. Once all commands for a drive have completed, all parameter blocks are checked to verify that they completed successfully.

4.5.1.14 Subtest 113, Verify Controller Reject

Subtest 113 first sends a head address outside the limit the controller is checking; the controller should reject it. Then the test sends a cylinder address outside the drive's range and a sector address outside the controller's limits. The controller should reject these as well.

4.5.2 Class 2 Subtests

Class 2 contains subtests that verify the drive's functionality. These tests verify that the drive can seek, switch heads, read, and reject a bad head or cylinder. The subtests contained in *dev4100* are shown in the following table.

Table 4–7, Class 2 Subtests

SUBTEST	DESCRIPTION	SMD REQUIRED	WRITE REQUIRED	TIME (min:sec)
200	Verify head switching	x	x	0:02
201	Verify sequential-track seeks	x	x	0:45
202	Perform minimum to maximum track seeks	x		0:10
203	Perform accordion seeks	x		1:15
204	Perform random seeks	x		1:15
205	Verify detect of forced faults (bad cylinder, head, and sector)	x		0:01

4.5.2.1 Subtest 200, Verify Head Switching

Subtest 200 verifies that the heads switch correctly. The test formats all heads on the diagnostic track and then reads them back with a *read track headers* command. It checks the head number in a header on each head and verifies the shift of the headers by one sector per head.

4.5.2.2 Subtest 201, Verify Sequential Track Seeks

Subtest 201, performs a *read track headers* command on each cylinder to confirm positioning.

4.5.2.3 Subtest 202, Perform Minimum to Maximum Track Seeks

Subtest 202 performs a *min-to-max* track alternate seek. Special formatting takes places for Class 2 subtests, if necessary, before the main portion of this subtest executes. The seek begins with an implied seek to the minimum track with a *read track headers* command. After the test verifies the cylinder number, a seek to the maximum track is performed using the *read track headers* command. When this cylinder is verified, the seek repeats to minimum and maximum tracks 100 more times with verification at each track. Each seek is an implied seek using the *read track headers* command. The seek then repeats, using the *seek* command with no verification until the seek is complete. To complete the test, the last track is read with a *read track headers* command and positioning is verified.

4.5.2.4 Subtest 203, Perform Accordion Seeks

Subtest 203 exercises the drive in an accordion seek. The seek starts with a *seek* to the minimum track, maximum track, min+1, max-1, min-2, max-2, etc., until a one-track seek is performed. The *read track headers* command actually issues an implied seek for each seek. The test verifies the seek by checking one of the headers just read. The accordion seek then repeats using the *seek* command and no verification. After the last seek is performed, a *read track headers* command is performed to verify that the final positioning is correct. The test chains commands for every drive, if allowed.

4.5.2.5 Subtest 204, Perform Random Seeks

Subtest 204 performs 1000 random seeks within the specified minimum and maximum cylinder. Random numbers are created by initializing C's random number generator with a seed value of 113 and then by calling *rand* for successive numbers. All seeks are implied by the *read track headers* command, which verifies positioning. The test then performs the random seek using the *seek* command with no verification until the seeks are complete. Finally, the test verifies the last cylinder with a *read track headers* command. The test chains commands for every drive, if allowed.

4.5.2.6 Subtest 205, Verify Detect of Forced Faults

Subtest 205 sends a *set drive size command,* which sets the head and cylinder limits to two greater than the maximum values specified in the file */mnt/bin/lib/DB_diskfmt.* Then the test commands the controller to read a track from a cylinder that is two greater than the maximum. This should result in a seek error. No error may occur if the number of cylinders specified in */mnt/bin/lib/DB_diskfmt* is less than the actual number of cylinders. In this case, an `Expected error did not occur` error is displayed.

Next, the test sets the head to maximum plus 2 and selects the diagnostic cylinder. This generates a write fault. After the write fault, the error does not clear until a drive reset is performed. To verify this, the test performs a read of head 0 of the diagnostic cylinder. A write fault should still occur. Finally, the test resets the drive and reads head 0 again; something other than a write fault should occur.

> **NOTE**
>
> A CDC 9766 drive will fail this test because it reports no error when an attempt to seek two cylinders beyond the defined maximum is made.

4.6 Disk Parameters File, *DB_diskfmt* Description

The disk parameters file, */mnt/bin/lib/DB_diskfmt*, contains information about disk drives. This information is required to be able to format/test new drives. Each line in the file contains a number of fields separated by one or more spaces. Comments start with a # in column 1. To add new drives, create a new entry with all the fields defined, and then add the drive to the */ioconfig* file. As of the time of this printing, *DB_diskfmt* contains the following information for all CONVEX machines:

Figure 4–5, Contents of the *DB_diskfmt* File

```
# DB_diskfmt - file of disk parameters
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#  Name          Description            Name          Description
# DKD-001       Fujitsu Eagle (452MB) DKD-008,208  NEC 2363 (1080MB)
# DKD-002       CDC 9766 (300MB)      DKD-214      Hitachi DK514-38 (356MB)
# DKD-005,206   NEC 2352 (500MB)
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a    b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-001 2   842 20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0   823 19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0   760 19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1 1024 27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a    b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-214 0   903 14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a    b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-206 0   760 19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0  1024 27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
# LEGEND:
# a - drive name                Must be DKD-0XX for Multibus and DKD-2XX for
#                               VMEbus
# b - disk type                 For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors     Number of actual sectors excluding runt
# f - # of logical sectors      Number of physical sectors (e) minus number of
#                               spares
# g - bits per sector           Number of bits between sector pulses
# h - bytes per track           Total number of unformatted bytes per track
# i - skew                      Sector offset from one head to the next
#                               Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks    .5% of number of cylinders (c). Raise
#                               fractional part to next higher whole number.
#                               Ignored by dev4110 (Multibus formatter)
# k - interleave                sector separation between consecutive sectors
#                               Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size                Number of halfwords in gap before header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# m - gap 2 size                Number of halfwords in gap following header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# n - drive interface           Used to determine how to read manufacturer's
#                               defect map.  Currently, smd or esdi
#                               Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme      Way data is encoded on the media. Used to
#                               select patterns for pattern test.
#                               Currently mfm, 2-7 or 1-7
# p - Are spares interleaved    For Xylogics, 'y'. For Interphase, 'n'.
```

4.7 Diagnostic Cylinder Description

The diagnostic cylinder is located on the first full cylinder preceding the sector forwarding area. Cylinder 840 is the diagnostic cylinder on a Fujitsu Eagle drive, and cylinder 820 is the diagnostic cylinder on a CDC SMD drive.

Each track of the cylinder is independent of the other tracks since there is a "table of contents" stored at sector 0 of each track This table describes what is stored on the remainder of that track. The format of this table is shown below:

Figure 4–6, Diagnostic Cylinder Table of Contents Format

```
/* ------------------------------------------------------------------ *
 *              Diagnostic Cylinder Definitions                       *
 * ------------------------------------------------------------------ */
#define NOT_USED        0x00000000        /* position in tbl not used     */
#define NO_HDR          0x00000001        /* this sectors header deleted  */
#define TBL_CONT        0x00000002        /* sector contains tbl of cntnts*/
#define ECC_ERR         0x00000003        /* sector written with bad ecc  */
#define PATTERN         0x00000004        /* sector written with pattern  */

 struct   tbl_cont {
    int   magic_nbr;            /* identifies this as a table of contents */
    int   version;              /* version number of this table */
    int   cyltksc;              /* sector header for this sector*/
    struct {
          int   sec_cont;         /* defines contents of corresponding sector */
          int   pattern;          /* pattern or mask used to verify contents  */
    } sec_tbl[62];
    int   checksum;             /* arithmetic tally of all the above fields */
 };
```

Each track on the cylinder is formatted in such a way that sector 0 is always on the track; sector 0 is written and verified without error when the drive is formatted.

The first field in the table contains the magic number 0x84101500. If this number is not present in the first position of a table, then someone has altered the diagnostic cylinder.

The second field in the table contains the version number. This number is an ordinal number starting with one and increasing as needed. Its purpose is to allow some measure of compatibility of newer diagnostic cylinders with older software. This number is checked if an unknown entry in the sector table (sec_tbl) is encountered. If the version number in the table is greater than the version number of the software, the diagnostic cylinder is assumed to have been formatted with a newer revision of software, and the opcode in the sector table is new to the older software. Therefore, the software is not familiar with the opcode, and the entry in the sec_tbl is ignored.

The next field in the table contains a copy of the sector header (cyltksc) for the table of contents.

The next field in the table contains the sector table (sec_tbl). This table describes what is stored on the rest of the sectors in this track. The sector table consists of 62 entries of 2 int's. Each entry (0 - 61) in this table, corresponds to a sector (0 - 61) on the track. The first position in

each entry is the sector contents (sec_cont) field. If the track has more than 62 sectors, the extra sectors are not used or checked by current software. The following table describes what is currently defined:

Table 4–8, Defined Values for Sector Contents Field

FIELD	DESCRIPTION
NOT_USED	This sector has not been initialized and should not be checked.
NO_HDR	When this sector is read, the controller should return a 0x05 sector not found error.
TBL_CONT	This sector contains a copy of the table of contents.
ECC_ERROR	This sector contains a copy of the table of contents that has been corrupted to give some form of ECC error. The pattern field contains a mask which may be xor'ed with the magic number field to correct the error. If the mask contains 11 or fewer one bits, the controller should report a soft ECC error when this sector is read. If the mask contains more than 11 one bits, the error reported should be a hard ECC error.
PATTERN	This sector has been filled with the data contained in the pattern field. No errors should be encountered when the sector is read.

The last field in the table of contents is a checksum. This is an arithmetic tally of all the bytes in the table of contents. This value is checked before any of the data in the table of contents is used.

4.8 Error Codes

The *dev4100* test reports controller and device errors in a standard format. However, if additional data is available at the time of the error, it reports that data also. The basic format for the errors is:

dev4100 ERROR (Err) ccu/mb/csr/d=c/m/rrr/d Err_desc

where:

(Err)	Identifies the error code associated with the description.
Err_desc	Describes the type of error that occurred.
c	Gives the CCU slot number for the IOP.
m	Is the Multibus chassis number.
rrr	Gives the control and status register address (CRS), which identifies the board.

d Specifies the device number. Each device begins at 0 with additional drives of the same type being numbered 1, 2, and 3. The number 4 indicates the controller itself was under test.

If no device is associated with the error, then the controller and device information does not appear in the error message.

Additional data may also be reported on a second line (depending on the type of error). For example, the format for data compare errors is:

Cyl:dddd Hd:dd

The following format appears when errors involve *write*, *read*, *read* and *write HDE*, *read* and *write track headers*, or *seek* commands (depending on the error, the sector number or number of seeks may not appear in the error message):

Cyl:dddd Hd:dd Sect:ddd Disp:ddd Exp:hhhh Act:hhhh #Seeks:d

where:

Cyl:dddd Specifies the position of heads on the drive.

Hd:dd Identifies the head selected.

Disp:ddd Identifies number of bytes from the beginning of the sector (data compare errors only).

Exp:hhhh Specifies the expected value (data compares errors only).

Act:hhhh Specifies the actual value (data compares errors only).

Sect:ddd Indicates the sector where the error occurred.

 # Seeks:d Identifies the number of seeks executed.

4.9 Error Messages

During execution, if the error limit is exceeded, the test aborts and displays:

```
Subtest Termination in Progress
```

For normal test completion, a message displays in the following format:

```
********** Test started Tue Mar 11 10:30:25 1986
********** Test ended   Tue Mar 11 10:34:15 1986

dev4100 Termination complete.
```

During execution, *dev4100* outputs error messages from the controller, IOP, disk device sequencer, and device subtests. The following sections describe these errors.

4.9.1 Controller Error Messages

> **NOTE**
>
> In the following error messages, IOPB represents Input Output Parameter Block.

0x00 IOPB completed without error
The controller completed the command successfully without an error.

0x01 Interrupt pending
The controller attempted an operation with a previous interrupt still pending. While an interrupt is pending, only the following operations are permitted: interrupt reset, update IOPB, controller reset, or error reset.

0x03 Attempt to write to reg. while busy
The IOP attempted a register write while controller busy is set. When the controller is busy, only bits 2 and 4 of the control and status register (CSR) have write access.

0x04 IOPB not completed within two seconds
The controller did not complete the input/output parameter block within two seconds.

0x05 Header not found
The controller cannot locate the requested sector. Any one of the following could cause this error:

- The actual number of physical sectors in the drive exceeds the maximum number of sectors plus 5. For example, if the controller searches 37 (32 + 5) sectors and the drive has 47 actual sectors, the controller may not compare 10 sectors for valid headers. (The controller compares headers for the maximum number of sectors plus 5.)
- The header the controller finds does not match the header ECC.
- The requested drive type and the drive type contained in the header do not match.
- A media defect may be in the header area.

0x06 Hard ECC error
This error only occurs on a *read* command when the controller detects a data error in the data field longer than 11 bits or when the ECC mode is disabled.

0x07 Bad cylinder from host
The IOP specified a cylinder address greater than the maximum cylinder number allowed.

0x0a Bad sector from host
This message indicates that the IOP specified a sector address greater than the maximum sector number allowed. Verify the maximum sector parameter for this drive type; retry the IOP operation.

0x0d Runt sector too small to write header
Either the last sector or all the sectors are too small to write a complete header. Verify the drive sector switches.

0x0e Memory failed to respond to DMA

This message means that the memory address by the controller failed to respond. The microprocessor provides a 10 millisecond timeout for the DMA sequencer to perform up to 128 transfers. When the timer interrupts, the controller verifies whether it is a bus master. A slave acknowledge error occurs if it is a bus master; a disk sequencer errors occurs when it is not a bus master. Verify the memory address or memory itself; retry the operation.

0x12 Header cylinder or head is incorrect

The cylinder or head address read from the disk does not match the IOPB cylinder and head address bytes.

This error occurs when the disk drive fails to seek to the correct cylinder. Another condition that causes this error is a corrupted disk format. Reformat the disk and retry the operation. In addition, if the head byte written on the disk does not match the selected head address, this error occurs. This indicates that there is a bad format or hardware problem.

0x13 Seek error was corrected by cntlr

The controller encountered a *seek* error. When the controller encounters a *seek* error, it automatically recalibrates the disk drive, clears the error, and completes the *seek*.

0x14 Write-protected disk

The controller attempted a *write* operation on a drive that is write-protected. First remove the write-protect and then retry the *write* operation.

0x16 Drive is not ready or faulted

This error indicates that the selected drive is not ready or possibly faulted. Any one of the following conditions can cause the problem:

- ALCO signal on the multibus backplane connector is low.
- Bad or improperly connected cable.
- Drive not up-to-speed or hardware error.
- Dual port access many not have been granted.
- No drive of the specified unit number is connected to the controller.

0x17 Bad sector count of 0 from host

This message indicates that the IOP issued the controller an IOPB with a sector count of zero. All data transfer operations require a positive sector count.

0x18 Drive reported a fault condition

This error messages indicates that a fault exists in the selected drive.

0x19 Bad sector size from host

The controller cannot write the header and data fields because the drive sectoring does not allow enough room. Any of the following conditions can cause this error to occur:

- The runt sector is too small.
- The drive contains more sectors than the number of specified data sectors plus five.
- Although the last sector is too small to be a data sector, it is included in the specified maximum sector. Either adjust the drive to include more sectors or the drive type (*DB_diskfmt*) to include fewer sectors.

0x1a Self-test A failed
Either the microprocessor or its internal RAM failed diagnostics.

0x1b Self-test B failed
This message indicates that either the microprocessor or header shift register failed diagnostics.

0x1c Self-test C failed
The buffer RAM failed diagnostics.

0x1e Correctable ECC error
The controller detected a correctable 11-bit (or less) error in the data field of the correct sector during a *read* in ECC mode 0.

0x1f Illegal ECC Mode used by Cntlr
During the transfer, the controller corrected one or more ECC errors in ECC mode 2, which should never be used as it does not work correctly in our configuration.

0x20 Bad head from host
The IOP specified a head address greater than the maximum head address allowed, which indicates a malfunctioning controller.

0x21 Disk sequencer error
This message indicates that the disk sequencer did not finish its operation within the allotted time. This can be caused by any of the following factors:

- The controller did not receive the servo clock signal from the selected drive. Check the cable connection.
- The controller is not receiving any read data from the selected drive. Check the cable.
- The Multibus may be preventing the controller from gaining proper access to memory.

0x25 Seek error
The IOP specified a cylinder address greater than the drive maximum or specified a head beyond that supported by the drive. Verify the drive parameters for the drive type tested.

0x3f Dual port drive already connected
The controller attempted to select a dual port drive which was connected to another controller. Since dual part drives are not supported, this error should never occur.

4.9.2 Device Subtest Error Messages

0x40 Bad block table is full
The bad block table can hold a maximum of 639 sectors. It is extremely unlikely that this table would ever fill up so it is much more likely that the bad block table is bad. In other words, the count of the number of used entries that is stored in the table is bad. This error sometimes occurs when a disk has been only partially formatted (headers are there and pattern-test data but no bad block table exists).

0x41 Bad sector not in Bad Block Table
A header not found occurred on a track which usually means the sector has been relocated. However, the sector was not found in the Bad Block Table. This may be due to a disk being only partially formatted.

0x42 No consecutive sectors found
The HDE subtest requires that two consecutive sectors exist on head 3 of the diagnostic cylinder. This error occurs if these headers are not found.

0x47 Bad parm to calc_cyl
This indicates a bug in the software. Please report it.

0x48 Bad parm to calc_hd
This indicates a bug in the software. Please report it.

0x49 Bad parm to calc_sec
This indicates a bug in the software. Please report it.

0x4a IOP print utility memory error
The print utility, prtlog_init, was unable to allocate main memory needed for prints from the CCU. Try running **mminit** and then retrying the test.

0x4b Device FAILED
The specified device failed this subtest. No more testing will be performed on this device.

0x4c Input parms don't allow write
The specified drive does not allow writes, and a write subtest was selected. The write is inhibited.

0x4d All copies of BBT are bad
There are five copies of the Bad Block Table on the last cylinder of each system-formatted disk. This error occurs when none of them are readable.

0x4e Diag. trk has all bad headers
During creation of the diagnostic cylinder, one of the tracks was found to be unusable because none of the sectors were readable. This error is very serious because it indicates either a major media flaw or a failure of that head to read data.

0x4f Diag. cylinder bad
During test of the diagnostic cylinder, sector zero of each track is read and two integer fields in the sector's data are checked against expected values. If they differ, this error results. This is usually caused by *dev4100* being run followed at some time later by a run of Subtest 103 in *dev4110*. The *dev4100* test writes over the diagnostics data written by Subtest 101 in *dev4110*. This problem is easily fixed by rerunning Subtest 101 in *dev4110*. Then use Subtest 103 in *dev4110* to verify the diagnostic cylinder is okay. If it still fails, then there is a problem writing or reading sectors.

0x50 Bad data compare on diag. cyl
Each track of the diagnostic cylinder has most of the sectors written with known patterns. These sectors are read and a data compare is performed to verify no data degradation or overwrite of this data has occurred. If any of these sectors do not successfully data compare, this error results. This could be caused by running *dev4100* prior to running Subtest 103 of *dev4110*.

0x51 N0 err. Hdr-not-found expected
One sector on each of the tracks of the diagnostic cylinder is deliberately removed so that a header not found error will result. If a header not found does not occur, this error is reported. This could be caused by running *dev4100* prior to running Subtest 103 of *dev4110*.

0x52 Soft ECC error did not occur
Subtest 103 of *dev4110* reports this error. Each track of the diagnostics cylinder has 11 sectors with soft ECC errors written to them. If any of these 11 sectors on each track do not cause the controller to report a soft ECC error, this error is reported.

0x53 Hard ECC error did not occur
Subtest 103 of *dev4110* reports this error. Each track of the diagnostics cylinder has one sector that has data with 12 consecutive bits in error. When this sector is read, it should always cause the controller to report a hard ECC error. If this error does not occur, this error is reported.

0x54 Data compare err on diag cyl
Most sectors on the diagnostic cylinder are written with known data patterns. During the test of the diagnostic cylinder, each sector containing known data is compared. This error occurs if a data compare fails. It may indicate degraded media or a bad head.

4.9.3 IOP Error Messages

0x80 Data compare error
Data read by the IOP did not match the expected data. The IOP performs the following data compares:
0x81 Seek requested with a seek count = 0
SEEK_CNT_ZERO – This indicates a bug in the software. Please report it.

0x82 Format of partial tracks not allowed
FORMAT_SETUP_BAD – This indicates a bug in the software. Please report it.

0x83 No attn acknowledge after attn request
NO_ATTN_ACK – This results when the controller is started on a chain of commands and then an attempt to get the attention of the controller in the middle of execution of the chain fails. This is a known problem with the controller and causes Subtest 112 to fail.

0x84 MBS error attempting to return msg
The IOP was attempting to send a message to the SPU when the MBS I/O subsystem reported an error condition.

0x85 Desired IOPB not in active chain
IOPB_NOT_IN_CHAIN – This is possibly due to a bad controller overwriting the pointer to the next IOPB in a chain. However, any component in the path of data from a controller to main memory is suspect.

0x86 Message received on wrong subqueue
The IOP received a message on a subqueue other than seven.

0x87 Wrong subqueue active and locked
The IOP detects a message on a subqueue other than seven, but the MBS locked the subqueue when a receive was attempted.

0x88 Wrong subqueue active and MBS error
The IOP detects a message on a subqueue other than seven. However, MBS reports an error when a message receive is attempted.

0x89 Wrong subqueue active, error invalid
The IOP detects a message on a subqueue other than seven. However, MBS reports an invalid error code when a message receive is attempted.

0x8a Driver executed for no reason
The Event Governed Operating System (EGOS) brought the IOP driver into execution; the driver found no reason for this action.

0x8b Unexpected soft error from Cntlr
The controller reported one of two soft errors when no soft error was expected. The error is one of the following:

Seek Retry Required	The controller detected a seek error and automatically recovered from the error. Because the bit that allows automatic recovery is not enabled, this error should never occur.
Soft ECC Error Recovered	The controller thinks it automatically corrected an ECC error of eleven bits or less. Because CONVEX does not use this function, this error should never occur.

0x8c Cntlr busy after hard error
After a hard error occurs, the controller goes "not busy" (according to the *Xylogics User's Manual*). In this case, a hard error was detected, and the controller was still busy when a retry was attempted.

0x8d Retry in progress-no active chain
NO_ACTIVE_CHAIN – This indicates a bug in the software. Please report it.

0x8e No attn acknowledge after attn request
Subtest 112 of *dev4100* verifies that, when a chain is actively executing, an attention request to the controller will cause the controller to halt without completing the chain and respond with attention acknowledge. This error indicates that this logic in the controller does not work properly. The Xylogics controller has this problem and will generate this error if Subtest 112 is enabled to run.

0x8f Cntlr not busy after release attn request
The controller is designed so that if busy is set when the attention request is asserted, busy remains set after the release of the attention request bit. If busy goes away, the controller does not meet specifications.

0x90 Cntlr busy after release attn
CNTLR_BUSY_ERR – The controller went not busy when during an attention request and before the controller was restarted. This is okay. However, when the controller's attention was released, the controller was found to be busy. This indicates a bad controller.

0x91 Interrupt occurred—no IOPBs done
The controller sent an interrupt; when the active IOPB chain was scanned, no completed IOPBs were found. This interrupt is not caused by an attention request, which is handled separately.

0x92 EGOS routine wndw_alloc returned 0
An attempt to set up a window to main memory from the IOP failed.

0x93 Task init attempt-error pending
RESPONSE_NEEDED – This indicates a bug in the software. Please report it.

0x94 No cntlr interrupt in 5 seconds
The controller must interrupt the driver within 5 seconds after an IOPB chain is started or a timeout occurs.

0x96 Write/Read error in controller regs
Write/Read test of controller registers failed.

0x97 Cntlr was busy after drive fault
The controller must go not busy after a drive fault. This is checked to ensure the controller is working as specified.

0x98 Cntlr stayed busy after drive reset
The controller is specified as going not busy after a drive reset. If it stays busy, this error is reported.

0x99 Attempting to poll-no active IOPBs
NOTHING_TO_POLL – This indicates a bug in the software. Please report it.

0x9a Finished task but int. state wrong
This indicates a bug in the software. Please report it.

0x9b Controller is not present
An attempt to write to the controller resulted in a BUS error, which implies that no controller is present or the controller is dead.

0x9c Sector interleave or skew error
The track headers were read to verify the format, and the headers were not properly interleaved or skewed.

0x9d Bad cylinder address after seek
A check of headers after *seek* revealed that the headers are on the wrong cylinder.

0x9e Set drive size - bad head or sector
A *read drive status* after setting the drive size in the controller showed that either the head or sector size was incorrectly set.

0x9f Set drive size - bad cylinder
A *read drive status* after setting the drive size in the controller showed that the cylinder size was incorrectly set.

0xa0 Expected error did not occur
Subtest 205 of *dev4100* selects a non-existent cylinder, then a nonexistent head is selected. An expected error had better occur or this error results.

0xa1 Error bit(s) set in cntlr csr
If a single or double error is reported in the command and status register and no other error is indicated, this error is reported.

0xa2 No free IOPBs for retry
NO_FREE_IOPBS – This indicates a bug in the software. Please report it.

4.9.4 Device Sequencer Error Messages

0xc0 MBS error-when SPU receiving a message
The MBS reported an error in attempting to provide the SPU with a message.

0xc1 Timeout because queue locked
The SPU received a message, but all ten attempts to read the message resulted in the error *MBS queue locked* being reported by the MBS.

0xc2 Msg interrupt but no msg found
The SPU attempted ten times to receive a message when an interrupt occurred, but all attempts failed with a *no message* error from the MBS.

0xc3 Bad return code from msg_build
BAD_BUILD_RTN – This indicates a bug in the software. Please report it.

0xc4 Timeout awaiting msg
SPU expected the IOP to send a message, but it did not receive a message. This error message indicates that the controller failed to generate an interrupt when done, or a break exists in the path of the interrupt signal, which prevents it from being detected.

0xc5 Rtn msgs from subq other than 7
The SPU received a message to the wrong subqueue; all messages are expected to arrive on subqueue seven.

0xc6 MBS returned bad error code
A call to an MBS routine returned an error code that was unrecognizable.

0xc7 Block transfer routine failed
A memory to memory transfer failed. Memory may be local or main.

0xc8 Too many devices failed. Failed subtest
This message indicates that the test is terminating because the device failure limit was exceeded.

0xc9 Return msg not of valid type
The SPU received a message, but it was not a device result or system error message.

0xca IOP's queue locked too long
Ten attempts to send a message to the IOP failed, with a locked condition being reported by MBS.

0xcb No room on SPU side for msg
Ten attempts to send a message failed because no messages were available for MBS to use.

0xcc Return msg error
An attempt to send a message from the SPU to the IOP resulted in an MBS error being reported.

0xcd Error while configuring I/O
The IOP returned a configuration message that specified a controller that does not exist.

0xce # active dev on cntlr < 0
BAD_DEV_CNT – This indicates a bug in the software. Please report it.

0xcf Too many controllers specified
The user attempted to select drives on too many controllers. This test's limit is 6. The limit for *dev4110* is 12.

0xd0 IOP echoed msgs when none sent
The SPU received an echo message, but no message had been sent.

0xd1 Bad cntlr # in returned msg
BAD_MSG – This indicates a bug in the software. Please report it.

0xd2 Outstanding msg cnt greater than 0
BAD_MSG_CNT – This indicates a bug in the software. Please report it.

0xd3 No msg err when sending to SPU
A message buffer was requested from MBS on the IOP. No message buffer was available. Try again later.

0xd4 MBS error when sending to SPU
The IOP MBS logic reported back an error condition (catchall for unexpected errors) while attempting to send a message to the SPU from the IOP.

0xd5 Inv. MBS err when sending to SPU
MBS on the IOP reported a non-zero error code that is not defined.

0xd6 MBS locked when sending to SPU
MBS on the IOP will not allow the sending of messages because the semaphore on the SPU process's queue is not being released.

0xd7 IOP driver device is bad
This indicates a bug in the software. Please report it.

0xd8 MBS error during IOP msg recv
The MBS routine that receives messages on the IOP reported that it detected an error. Usually this means that main memory no longer contains valid message queues.

0xd9 MBS queue locked during receive
The SPU attempted to receive a message but its queue was locked too long. Should never happen.

0xda MBS no message available
A message should have been in the queue but when it was checked, there were no messages.

0xdd Processor queue setup failed
PQ_SYSTEM_FAIL – The SPU utility, **pqutil** was unable to initialize the processor queues which are needed for SPU<->CCU communications. This is usually due to main memory not being initialized. Try running **mminit** before the test.

0xde Failed to open window
WNDW_OPEN_ERR – An attempt to open the file */dev/wndw* on the SPU disk failed. See if the file exists and verify it has global write permission.

0xdf Failed to allocate map registers
IOCTL_ERR – An attempt to set up map registers to main memory for access from the SPU has failed. This should never happen. Please report it.

0xe0 IOP load module not found
The file that is to be loaded into the IOP does not exist. The current directory is searched first and then */mnt/test* is searched.

0xe1 Error attempting to load iop
The OS utility */mnt/os/loadccu* returned non-zero when it attempted to load the IOP. The usual cause is that *mminit* or a *sysreset* was not performed prior to running the test.

0xe2 Main memory allocation error
MMALLOC_ERR – An attempt to allocate main memory has failed. Try running **mminit** before rerunning the test.

Appendix A
Reporting Problems

A.1 Overview

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

A.2 Technical Assistance Center

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

A.3 The *contact* Utility

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

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

A.4 Prerequisites

To use *contact* requires

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

A.4.1 UUCP Connection

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

A.4.2 Finding the Program Path Name

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

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

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

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt. If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, only faster.

A.4.3 Finding the Program Version Number

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

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

In this example, the loader utility version number is 7.0.

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

A.5 Tips on Using the *contact* Utility

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

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

A.5.1 Using a *.contact* File

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

1. Create a *.contact* file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

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

A.5.2 Aborting the Report

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

A.5.3 Submitting the *dead.report* File

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

contact -r

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

A.5.4 Suspending a Report

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

A.5.5 Ending a Response

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

A.5.6 Tilde-Escape Sequences

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

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

A.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

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

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

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

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

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

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

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

Enter the name of the product involved
>
```

2. The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if using *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.
3. The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Make this summary as descriptive as possible in one line.
4. The *contact* utility prompts for a detailed description of the problem. Make this description as complete as possible. Include source code and a stack backtrace whenever possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information provided, the quicker the TAC can isolate and solve the problem.
5. The *contact* utility prompts for the priority of the problem. The following screen illustrates this prompt and the priority levels from which to choose; you must enter a priority number.

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

6. The *contact* utility prompts for an explanation of how to reproduce the problem. Include the command syntax and options you used and anything else you did to make your program run.
7. The *contact* utility prompts for any other pertinent comments. Include any relevant information.
8. The *contact* utility prompts for suggestions regarding the documentation supporting the product. Indicate if the documentation could be revised to address the question.
9. The *contact* utility asks for the names of files necessary to reproduce the problem. The following screen illustrates the *contact* prompt and sample user response:

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

NOTE

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

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

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

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

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

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

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

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

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

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

Index

R

S

T

U

V

W

X

CONVEX Multibus Storage Module Drive (SMD)
Disk (*dev4100*) Diagnostics Manual
Document No. 760-001930-000
First Edition

Reader's Forum

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska & Hawaii	1(214)497-4379
From locations in Canada	1(800)345-2384
From all other locations	Contact nearest CONVEX office

Direct mail orders to:

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

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

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

(Fold Here Second)

(Tape or Staple)