CONVEX Multibus STC Tape Unit Controller (*dev4200*) Diagnostics Manual

Document No. 760-002130-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus STC Tape Unit Controller
(dev4200) Diagnostics Manual
Order No. DHW-233
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus STC Tape Unit Controller (dev4200) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002130-000	May 1991	First release. Contains the *dev4200* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Diagnostics Environment

1.1 Overview 1-1
1.2 Test Program Naming Conventions 1-1
1.2.1 Test Program Categories 1-1
1.2.2 Test Program Types 1-2
1.2.3 Test Program Device Types 1-2
1.2.4 Examples of Test Program Names 1-3

2 EGOS Overview

2.1 Overview 2-1
2.2 Purpose of EGOS for Diagnostic Testing 2-1
2.3 EGOS for the Multibus Interface 2-1
2.4 EGOS for HSP Interface, HSP EGOS 2-1
2.5 EGOS for VME Interface, VIOP EGOS 2-2
2.6 EGOS Position in the Environment 2-2

3 Dshell Overview

3.1 Overview 3-1
3.2 Diagnostic Shell (*dshell*) Overview 3-1
3.3 Syntax Help for *dshell* Commands 3-3

4 Multibus STC Tape Unit Controller Test (*dev4200*)

4.1 Overview 4-1
4.2 Prerequisites and Required Equipment 4-1
4.3 Test Invocation 4-1
4.3.1 Test Parameter Menu 4-3
4.3.2 Prompt Explanations 4-5
4.4 Hardware Initialization Sequence 4-7
4.5 Class Descriptions 4-7
4.5.1 Class 1 Subtests 4-8
4.5.1.1 Subtest 100, MBTC Reset Capability 4-8
4.5.1.2 Subtest 101, MBTC Interrupt Loopback 4-9
4.5.1.3 Subtest 102, MBTC Data Loopback 4-9
4.5.1.4 Subtest 103, MBTC Memory Access 4-9
4.5.1.5 Subtest 104, MBTC Command Loopback 4-9
4.5.1.6 Subtest 105, MBTC Chain Command Loopback 4-9
4.5.2 Class 2 Subtests 4-9
4.5.2.1 Subtest 200, MBTC Drive Online 4-9
4.5.2.2 Subtest 201, MBTC Drive Write Protect 4-10
4.5.2.3 Subtest 202, MBTC Drive Unload 4-10
4.5.3 Class 3, 4, and 5 Subtests 4-10
4.5.3.1 Subtest n00, MBTC Write Tape Mark 4-11
4.5.3.2 Subtest n01, MBTC File Skip Forward 4-11
4.5.3.3 Subtest n02, MBTC File Skip Backward 4-12
4.5.3.4 Subtest n03, MBTC Read File Marks Forward 4-13
4.5.3.5 Subtest n04, MBTC Read File Marks Backward 4-13
4.5.3.6 Subtest n05, MBTC Block Skip File Marks Forward 4-13
4.5.3.7 Subtest n05, MBTC Block Skip File Marks Backward 4-13
4.5.3.8 Subtest n10, MBTC Write Forward 4-13

4.5.3.9 Subtest n11, MBTC Read Forward 4-13
4.5.3.10 Subtest n12, MBTC Read Backward 4-13
4.5.3.11 Subtest n13, MBTC Block Skip Forward 4-13
4.5.3.12 Subtest n14, MBTC Block Skip Backward 4-14
4.5.3.13 Subtest n20, MBTC Erase Gap 4-14
4.5.3.14 Subtest n25, MBTC End-of-Tape Sensing 4-14
4.5.3.15 Subtest n30, MBTC Force Parity Error 4-14
4.5.3.16 Subtest n50, MBTC Write Files with Fixed Length Records 4-14
4.5.3.17 Subtest n51, MBTC Read Files with Fixed Length Records 4-15
4.5.3.18 Subtest n52, MBTC Write Files with Variable Length Records 4-15
4.5.3.19 Subtest n53, MBTC Read Files with Variable Length Records 4-15
4.5.3.20 Subtest n60, MBTC Write Chained Mode Data Files 4-15
4.5.3.21 Subtest n61, MBTC Read Chained Mode Data File 4-16
4.6 Error Messages 4-16

Appendixes

A Reporting Problems

A.1 Overview A-1
A.2 Technical Assistance Center A-1
A.3 The *contact* Utility A-1
A.4 Prerequisites A-1
A.4.1 UUCP Connection A-1
A.4.2 Finding the Program Path Name A-2
A.4.3 Finding the Program Version Number A-2
A.5 Tips on Using the *contact* Utility A-2
A.5.1 Using a *.contact* File A-3
A.5.2 Aborting the Report A-3
A.5.3 Submitting the *dead.report* File A-3
A.5.4 Suspending a Report A-3
A.5.5 Ending a Response A-3
A.5.6 Tilde-Escape Sequences A-4
A.6 Using the *contact* Utility A-4

List of Tables

1–1 Test Program Categories 1-2
1–2 Test Program Types 1-2
1–3 Test Program Device Types 1-3
1–4 Example Test Program Names 1-3
3–1 *dshell* Commands 3-2
4–1 Hardware Requirements 4-1
4–2 Getting Help During Test Parameter Entry 4-3
4–3 *dev4200* Test Classes 4-8
4–4 Class 1 Subtests 4-8
4–5 Class 2 Subtests 4-9
4–6 Class 3, 4, and 5 Subtests 4-10
4–7 Execution Times for $n = 4$ 4-11
4–8 *setup_iop* Exception Error Messages 4-16

4–9 IOP Processing Error Messages 4-17

List of Figures

2–1 EGOS' Position in the Environment 2-3
3–1 Syntax Help for the *loop* Command 3-3
4–1 Test Invocation Sequence 4-2
4–2 Alternate Test Invocation Sequence 4-3
4–3 Test Parameter Menu 4-4
4–4 Sample Test Parameter Summary 4-7

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4200* diagnostic, which verifies the functionality of CONVEX Multibus STC tape controllers. This document is not a tutorial, but rather a reference for the users of the *dev4200* 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 *dev4200* 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 STC Tape Unit Controller Test (*dev4200*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes the error messages produced by the diagnostic.
- **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-233.
The document number for this manual is 760-002130-000.

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

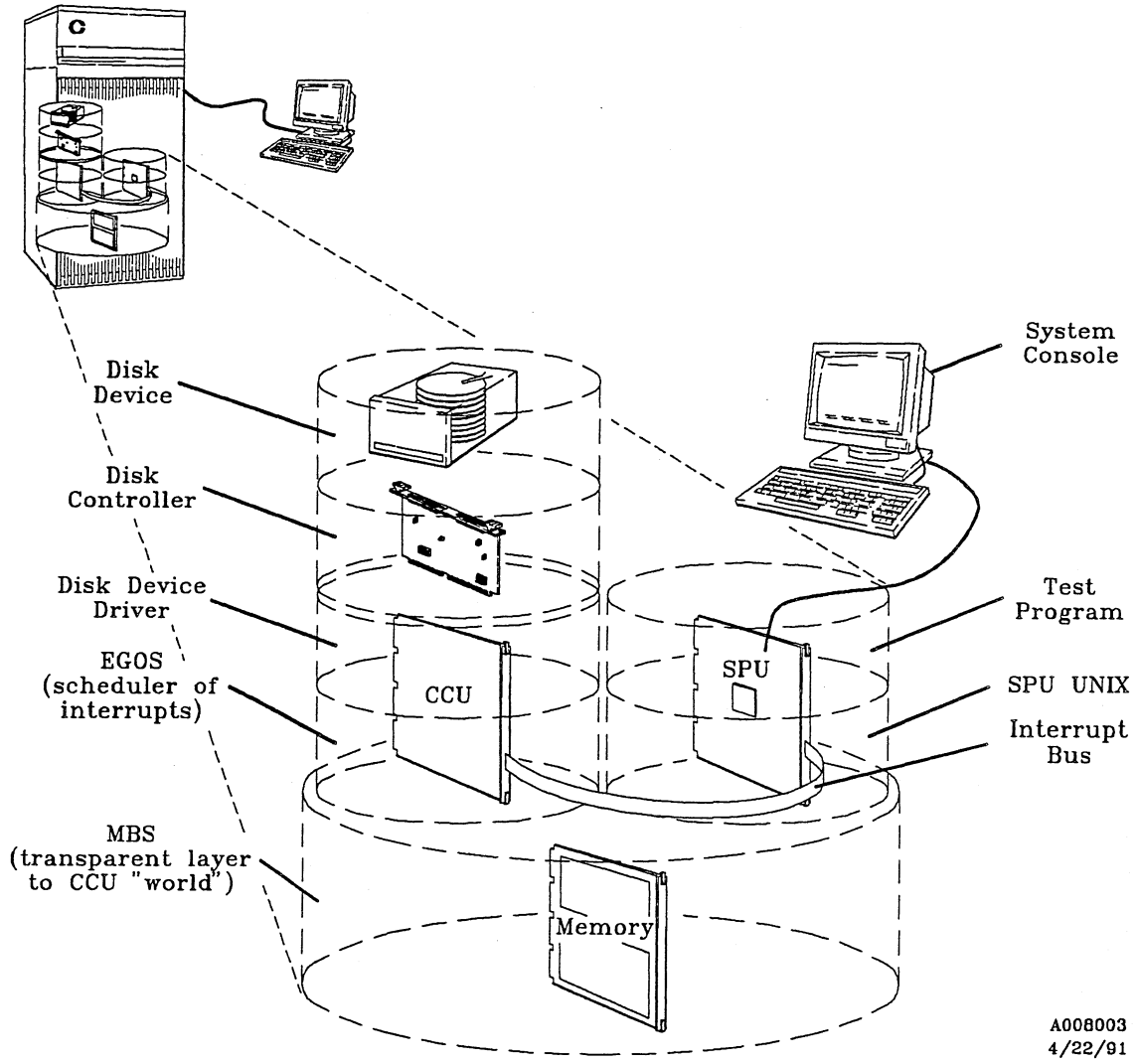
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes 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 STC Tape Unit Controller Test (*dev4200*)

4.1 Overview

The *dev4200* test verifies the functionality of CONVEX Multibus STC tape controllers.

4.2 Prerequisites and Required Equipment

The following table lists the required hardware depending on the type of machine under test.

Table 4–1, Hardware Requirements

C1, C120	C200 Series
MCU	Memory System[1]
MAU	CPX
SPU	SP2
IOP	IOP
MBCU	PIA
MBTC[2]	MBCU
Tape Drive	MBTC[2]
	Tape Drive

[1] Memory System consists of a minimum of one pair of memory boards (one odd and one even).

[2] MBTC represents Multibus Tape Controller

4.3 Test Invocation

The *dev4200* 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 *dev4200* 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 dev4200 [-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 dev4200** executes all *dev4200* 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 dev4200 [-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.

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

Table 4–2, Getting Help During Test Parameter Entry

Character	Description
?	Displays this help menu
h	Provides help for a specific prompt
i	Displays the */ioconfig* file

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

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

                PERIPHERAL CONFIGURATION DATA
       CCU      Chassis Type   CSR     Int     Unit    Type
       ------   ------- ------- -----   ---     ----    -------
 1)    iop 3      0     MTC-001 0x0c0    4       0     MTD-001
 2)    iop 3      0     MTC-001 0x0c0    4       0     MTD-001

Device 0 = user defined configuration

 1: Device Selection [0,1-2][1]                        (0) ->
 2: IOP [3-7][2]                                      (5) ->
 3: Multibus Chassis [0-3]                            (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]
                                                   (0xc0) ->
 5: Controller Interrupt [0-7]                        (4) ->
 6: Tape Unit [0-3]                                   (0) ->

        Tape Unit Types:
        0 -> STC 2920
        1 -> STC 1960
        2 -> FUJITSU M2436
        3 -> CDC 92185-04

 7: Tape Unit Type [0-3]                              (0) ->
 8: Media Error Retry Count in Subtests n50-n53 [0-15]
                                                      (5) ->
 9: Display Each Media Error [y,n]                    (n) ->
10: Run Subtests n50-n53 to EOT [y,n]                 (n) ->
11: Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
    [18-131071]                                    (512) ->
12: Run EOT Subtest n25 [y,n]                         (n) ->
13: Allow Manual Intervention [y,n]                   (n) ->
14: Enter OK, or :NN to return to question NN [OK]
                                                     (OK) ->
```

[1] The options for this prompt change depending on how many tape units are in the system configuration.

[2] The options for this prompt change depending on the machine architecture. For C1 and C120 machines, the available selection ranges from 3-7. For C200 Series machines, the available selection ranges from 0-3.

> **NOTE**
>
> In prompts 8, 10, 11, and 12 in the previous figure, *n* varies from 3 to 5 depending on which classes are executed. Class 3 subtests execute at 800 bpi, Class 4 subtests execute at 1600 bpi, and Class 5 subtests execute at 6250 bpi. Refer to class and subtest descriptions for more information.

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.

```
Device Selection [0,1-2]                    (0) ->
```

This prompt selects a device for testing from the system configuration file (*/ioconfig*). If **0** (default) is entered, then user-defined configuration prompts (2-5 in the previous figure) are displayed to determine the configuration to be tested. If a specific device is to be selected, the number of the device is entered (**1**, **2**, etc...). At that point, the additional configuration prompts (2-5 in the previous figure) are not displayed.

```
IOP [3-7]                                   (5) ->
```

Enter the CCU slot number for the desired IOP.

```
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]
                                          (0xc0) ->
```

Enter the low-order 12 bits of the controller's address within the Multibus (this address is selected with switches on the controller).

```
Controller Interrupt [0-7]                  (4) ->
```

Enter the interrupt level of the controller within the Multibus (this is selected with switches on the controller).

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

Enter the unit number of the tape unit to be tested.

```
Tape Unit Type [0-3]                               (0) ->
```

Enter the number of the type of tape unit to be tested.

```
Media Error Retry Count in Subtests n50-n53 [0-15]
                                                   (5) ->
```

Enter the number of retries to be performed when a media error is detected during Subtests n50–n53.

```
Display Each Media Error [y,n]                     (n) ->
```

Enter **n** or RETURN so media errors and retries are not displayed during all subtests; otherwise, enter **y** to display all media errors and retries during all subtests.

```
Run Subtests n50-n53 to EOT [y,n]                  (n) ->
```

Enter **y** to write as many files of data as are required to the end of the tape during Subtests n50–n53.

```
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
[18-131071]                                      (512) ->
```

Enter RETURN to select 512 bytes as the block size of records written to the tape during Subtests n10–n12, n30, n50, n51; otherwise, enter the block size (in bytes) of records written to the tape.

```
Run EOT Subtest n25 [y,n]                          (n) ->
```

Enter **n** or RETURN to skip Subtest n25; otherwise, enter **y** to allow execution of Subtest n25, End-of-Tape Sensing.

```
Allow Manual Intervention [y,n]                    (n) ->
```

Enter **n** to skip Subtests 104, 105, 200–202; otherwise, enter **y** to allow execution of Subtests 104, 105, 200–202.

```
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.

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

Device Selection                                        : 0
IOP                                                     : 5
Multibus Chassis                                        : 0
Controller Offset in Multibus                           : 0xc0
Controller Interrupt                                    : 4
Tape Unit                                               : 0

        Tape Unit Types:

            0 -> STC 2920
            1 -> STC 1960
            2 -> FUJITSU M2436
            3 -> CDC 92185-04

Tape Unit Type                                          : 0
Media Error Retry Count in Subtests n50-n53             : 5
Display Each Media Error                                : n
Run Subtest n50-n53 to EOT                              : n
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
                                                        : 512
Run EOT Subtest n25                                     : n
Allow Manual Intervention                               : n
Enter OK, or :NN to return to question NN              : OK
```

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

The *dev4200* test contains the five classes of subtests listed in the following table:

Table 4–3, *dev4200* Test Classes

CLASS	DESCRIPTION
1	Controller loopback tests
2	Operator controls tests
3	Functional tests—800 bpi
4	Functional tests—1600 bpi
5	Functional tests—6250 bpi

All of the subtests contained in *dev4200* are loopable under the *dshell*.

4.5.1 Class 1 Subtests

Class 1 subtests verify the basic functionality of the controller. The subtests verify board reset and loopback capabilities. By entering **n** (the default) for the `Allow Manual Intervention` test parameter prompt, Subtests 104 and 105, which require manual intervention are not executed.

Table 4–4, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	MBTC Reset Capability	00:01
101	MBTC Interrupt Loopback	00:05
102	MBTC Data Loopback	00:25
103	MBTC Memory Access	04:30
104	MBTC Command Loopback	00:10
105	MBTC Chain Command Loopback	00:10

The execution times in the previous table are approximate times for testing a STC 2920 drive at 1600 BPI. Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times include rewind time from the tape position at the end of the last subtest.

The *dev4200* test does not test the drive commands: *DMS*, *SNS*, and *LWR*. In addition, it does not test the following controller-error handling capabilities:

- Pending command abort
- Multibus timeout

Also, certain drive capabilities are not tested. This includes all command reject and abort scenarios.

4.5.1.1 Subtest 100, MBTC Reset Capability

Subtest 100 verifies that the controller resets properly by checking the status register for the correct value.

4.5.1.2 Subtest 101, MBTC Interrupt Loopback

Subtest 101 verifies that the board interrupt capability can be turned off and on.

4.5.1.3 Subtest 102, MBTC Data Loopback

Subtest 102 tests the direct memory access First-In-First-Out (FIFO) memory on the board. It pattern checks all RAM's in the FIFO and tests in/out capability, assuming the FIFO operates like a circular buffer with pointers.

4.5.1.4 Subtest 103, MBTC Memory Access

Subtest 103 checks the capability of the board to access all allowed pages in main memory via the DMA and associated buffers.

4.5.1.5 Subtest 104, MBTC Command Loopback

Subtest 104 checks the capability of the board to properly register command status for the executing and pending states. This subtest does not execute when **n** (the default) is entered selected for the test parameter prompt `Allow Manual Intervention`.

4.5.1.6 Subtest 105, MBTC Chain Command Loopback

Subtest 105 checks the capability of the board to properly register command status while executing chaining commands. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

4.5.2 Class 2 Subtests

Class 2 subtests verify operator controls, tape write protect capability, and unload reel capability. All subtests in this class are skipped if **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

Table 4–5, Class 2 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
200	MBTC Drive Online	Manual intervention time
201	MBTC Drive Write Protect	Manual intervention time
202	MBTC Drive Unload	Manual intervention time

4.5.2.1 Subtest 200, MBTC Drive Online

Subtest 200 verifies that drive online status is properly reflected to the software as the operator manipulates the drive controls. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

4.5.2.2 Subtest 201, MBTC Drive Write Protect

Subtest 201 verifies that file protection is properly reflected to the software as the operator removes and reinserts the write ring. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

4.5.2.3 Subtest 202, MBTC Drive Unload

Subtest 202 tests the drive unload operation. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

4.5.3 Class 3, 4, and 5 Subtests

Classes 3, 4, and 5 contain subtests to verify correct operation of the controller and drive for a wide selection of commands and data transfer scenarios, including chaining mode. These subtest are in the range of 300-599 (n00-n99). Subtest numbers in these three classes contain a prefix of *n* (as the hundreds digit), which is used to select tape density. The 300 series subtests are for 800 bpi tapes, 400 series for 1600 bpi, and 500 series for 6250 bpi.

Table 4–6, Class 3, 4, and 5 Subtests

SUBTEST	DESCRIPTION
n00[1]	MBTC Write Tape Mark
n01	MBTC File Skip Forward
n02	MBTC File Skip Backward
n03	MBTC Read File Marks Forward
n04	MBTC Read File Marks Backward
n05	MBTC Block Skip File Marks Forward
n06	MBTC Block Skip File Marks Backward
n10	MBTC Write Forward
n11	MBTC Read Forward
n12	MBTC Read Backward
n13	MBTC Block Skip Forward
n14	MBTC Block Skip Backward
n20[1]	MBTC Erase Gap
n25	MBTC End-of-tape Sensing
n30	MBTC Forced Parity Error
n50	MBTC Write Files with Fixed Length Records
n51	MBTC Read Files with Fixed Length Records
n52	MBTC Write Files with Variable Length Records
n53	MBTC Read Files with Variable Length Records
n60	MBTC Write Chained Mode Data Files
n61	MBTC Read Chained Mode Data File

[1] *n* varies from 3 to 5; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

The following execution times, which are for an STC 2920 drive at 1600 BPI, are approximate. Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times except *n*00 include rewind time from the tape position at the end of the last subtest. If subtests *n*50-*n*53 are run to the end of the tape, then subtest times are appropriately lengthened, depending on the length of the tape.

Table 4–7, Execution Times for $n = 4$

SUBTEST	TIMES (min/sec)
400[1]	00:10
401	00:15
402	00:20
403	00:10
404	00:20
405	00:10
406	00:20
410	00:10
411	00:15
412	00:20
413	00:10
414	00:15
420	00:10
425	time to end of tape
430	01:15
450	01:15
451	02:00
452	02:15
453	03:20
460	02:55
461	02:45

[1] Execution times for the 300 and 500 series subtests vary from those shown for the 400 series.

4.5.3.1 Subtest n00, MBTC Write Tape Mark

Subtest *n*00 rewinds the tape and then writes a pattern of 100 tape marks. Media errors are ignored unless the count exceeds 5.

> **NOTE**
>
> In this subtest and all the following subtests, *n* varies from 3 to 5 based on tape density; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

4.5.3.2 Subtest n01, MBTC File Skip Forward

Subtest *n*01 rewinds the tape and then uses the file *skip forward* function to skip 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

4.5.3.3 Subtest n02, MBTC File Skip Backward

Subtest *n*02 rewinds the tape, skips 90 tape marks written by Subtest *n*00 in the forward direction, and then skips 80 tape marks in the reverse direction. This subtest ignores media errors unless the count exceeds 10 for tape motion in either direction.

4.5.3.4 Subtest n03, MBTC Read File Marks Forward

Subtest *n*03 rewinds the tape, and then reads 90 tape marks written by Subtest *n*00 with a *block read* function in the forward direction. Media errors are ignored unless the count exceeds 10.

4.5.3.5 Subtest n04, MBTC Read File Marks Backward

Subtest *n*04 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and reads 80 tape marks backward, using the *block read backward* function. This subtest ignores media errors unless the count exceeds 10 in either direction.

4.5.3.6 Subtest n05, MBTC Block Skip File Marks Forward

Subtest *n*05 rewinds the tape and then uses the *block skip forward* function to skip over 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

4.5.3.7 Subtest n05, MBTC Block Skip File Marks Backward

Subtest *n*06 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and then it skips 80 tape marks backward using the *block skip backward* function. The subtest ignores media errors unless the count exceeds 10 in either direction.

4.5.3.8 Subtest n10, MBTC Write Forward

Subtest *n*10 rewinds the tape and then writes a data pattern composed of 200 records, each containing incrementing byte values, modulo 256, beginning with zero, followed by 5 tape marks. The number of bytes entered at the test parameter prompt `Fixed Length Block Size` determines the record size; the default record size is 512 bytes. Media errors and block size errors are ignored unless the total count exceeds 10. Errors writing the tape marks are ignored if at least one is valid.

4.5.3.9 Subtest n11, MBTC Read Forward

Subtest *n*11 first rewinds the tape and then reads forward 190 records written by Subtest *n*10. This subtest ignores media, block size, and data errors unless the total error count exceeds 10.

4.5.3.10 Subtest n12, MBTC Read Backward

Subtest *n*12 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip* function, and then the pattern is read backward for 190 records. The subtest ignores media, block size, and data errors unless the total error count exceeds 10.

4.5.3.11 Subtest n13, MBTC Block Skip Forward

Subtest *n*13 rewinds the tape and then skips over 190 tape blocks written by Subtest *n*10 using the *block skip* function. Media errors are ignored unless the error count exceeds 10.

4.5.3.12 Subtest n14, MBTC Block Skip Backward

Subtest *n*14 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip forward* function, and then it skips 190 tape blocks using the *block skip backward* function. Media errors are ignored unless the error count exceeds 10.

4.5.3.13 Subtest n20, MBTC Erase Gap

Subtest *n*20 rewinds the tape, performs at least 50 erase-gap functions, and then performs a *block skip backward* function. The *block skip backward* function is timed to assure that the long gap exists. The subtest ignores media errors when erasing unless the count exceeds 5.

4.5.3.14 Subtest n25, MBTC End-of-Tape Sensing

Subtest *n*25 rewinds the tape and performs a *write forward* operation until the EOT marker is read. This subtest executes only when **y** to the `Run EOT Subtest n25` test parameter prompt.

4.5.3.15 Subtest n30, MBTC Force Parity Error

Subtest *n*30 rewinds the tape and writes a data pattern with the controller set to force parity errors. The forced errors are verified. The subtest ignores media and block size errors unless the total count exceeds 10.

Then the subtest rewinds the tape and a valid data pattern is written without forced errors. Media and block size errors are ignored unless the total count exceeds 10. Then the subtest rewinds the tape again and reads with forced parity errors. The subtest ignores media, block size, and data errors unless the total count exceeds 10.

4.5.3.16 Subtest n50, MBTC Write Files with Fixed Length Records

Subtest *n*50 rewinds the tape, and then writes 10 files of data patterns. Each file consists of 200 fixed-length data blocks and is terminated with a tape mark. A second tape mark follows the last file. Each data block, which has its length determined by the number entered for the test parameter prompt `Fixed Length Block Size` (default 512 bytes), consists of increasing byte values, modulo 256, beginning with the block number (modulo 256) in the file. The first byte in each block is an exception; it contains the alternating values 0xaa and 0x55. When the end of tape marker is read the first byte contains the value 0xff.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (the default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with Subtest *n*51. Entering **y** to the test parameter prompt `Run Subtests n50-n53 to EOT` causes the subtest to write as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

4.5.3.17 Subtest n51, MBTC Read Files with Fixed Length Records

Subtest *n*51 rewinds the tape and then reads and verifies the data structure written by Subtest *n*50. The subtest recovers media errors by using a backspace plus reread technique. Any media error is retried the number of times specified for the test parameter prompt `Media Error Retry Count` (default is 5). The subtest detects the end of tape (if any) by reading software marks written by Subtest*n*50.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

4.5.3.18 Subtest n52, MBTC Write Files with Variable Length Records

Subtest *n*52 rewinds the tape and then writes one file to the tape. The file is terminated with two tape marks. The data in the file consists of records of various lengths, from 18 to 128*1024-1 bytes. Each record contains increasing byte values, modulo 256, beginning with the block size. The first byte in each block is the end of tape marker, which normally has a value of 0xaa. The first byte has a value of zero after the end of tape marker is read.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with the read Subtest *n*53. If **y** is entered to the test parameter prompt `Run Subtests n50-n53 to EOT`, the subtest writes as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

4.5.3.19 Subtest n53, MBTC Read Files with Variable Length Records

Subtest *n*53 rewinds the tape and then reads and verifies the data structure written by Subtest *n*52. The subtest recovers media errors by using a backspace plus reread technique. The subtest detects the end of tape by reading software marks (if any) written by Subtest *n*52.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

4.5.3.20 Subtest n60, MBTC Write Chained Mode Data Files

Subtest *n*60 rewinds the tape and then writes one file to the tape. The file consists of several records of various sizes written in chained mode. Two file marks follow the data. The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering **y** to the `Display Each Media Error` test parameter prompt.

4.5.3.21 Subtest n61, MBTC Read Chained Mode Data File

Subtest *n*61 rewinds the tape and then reads and verifies the data structure written by Subtest *n*60. The subtest recovers media errors by using a backspace plus reread technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering **y** to the `Display Each Media Error` test parameter prompt.

4.6 Error Messages

Erase Gap Failed
The gap erase function did not succeed in erasing a gap on the tape.

Exception for iop d from recv_iop {error message detail}
An error has occurred using the SPU/IOP interface to wait for a signal from the IOP that the command is finished. Refer to "Exception from setup_iop" for message detail.

Exception from load_iop {error message detail}
An error has occurred while bootstrapping the IOP. Make sure that the IOP program image file *dev4200.x00* is available and readable. It must reside either in the current directory or */mnt/test*. Refer to "Exception from setup_iop" for message detail.

Exception from send_iop (*n*) {error message detail}
An error has occurred using the SPU/IOP interface to signal the IOP that a command is ready. Refer to "Exception from setup_iop" for message detail.

Exception from setup_iop {error message detail}
An error occurred when preparing to access the IOP. When this error occurs, the message detail may be:

Table 4–8, *setup_iop* Exception Error Messages

TEXT	MEANING
HARD ERROR	Hardware error
IOP BUS ERROR	Controller address error
IOP CACHE ERROR	IOP hardware error
IOP PBUS ERROR	IOP hardware error
MMIO ERROR	Main memory error
MULTIBUS ERROR	Hardware error
TIMEOUT	Possible hardware error

Exception from start_iop {error message detail}
An error has occurred while starting the IOP. Refer to "Exception from setup_iop" for message detail.

Exception in mmalloc_init. Is main memory initialized?
There is a problem accessing main memory. Assure that main memory is present; initialize, if necessary, using *mminit*. If problems persist, consult the Technical Assistance Center.

Function status: message {UNIX error message or tape drive status dump, if applicable}
There is a problem in IOP processing. The following messages may appear:

Table 4–9, IOP Processing Error Messages

TEXT	MEANING
TIME OUT	no response to device command
CHAINING OVERRUN	program and tape drive got out of sync chaining
MEDIA ERROR	tape error

Main memory allocation error
There is a problem allocating main memory. Be sure that main memory is present and initialized. If problems persist, consult the Technical Assistance Center.

MBTC exec level command: *"ccccccccc" (0xnn)*
This message gives a description of the command, where *ccccccccc* is the verbal description of the command.

MBTC exec level command: expected *"ccccccccc" (0xnn)* **actual** *"ccccccccc" (0xnn)*
This message gives the expected and actual command, where *ccccccccc* is the verbal description of the command.

MBTC exec level unit: *nn*
This message gives the unit number in the execution level of the controller.

MBTC exec level unit: expected *nn*, **actual** *nn*
This message indicates a discrepancy in the unit number in the execution level of the controller.

MBTC cstat[1] *0xnn* **means:**
{error message detail}
This indicates a controller error status byte at offset 0x18 of controller. (The error message detail appears in the next entry.)

MBTC cstat[1]: expected *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}
This indicates a discrepancy in the controller status at offset 0x18. The interpretation may be:

```
LAST BYTE [IN]VALID
[NO] MULTIBUS DMA TIMEOUT
[NO] READ PARITY ERROR
[NO] TAPE MARK
PENDING COMMAND [NOT] ABORTED
TAPE ERROR SUMMARY BIT [NOT] SET
```

MBTC cstat[2] *0xnn* **means:**
{error message detail}
This gives the interpretation of the formatter status byte at controller offset 0x19. (Possible interpretations appear in the next message entry.)

MBTC cstat[2]: expected *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}
This message indicates a discrepancy in the formatter status byte at offset 0x19. The error message detail may be:

```
FORMATTER COMMAND [IN]COMPLETE
FORMATTER COMMAND REJECTED
FORMATTER COMMAND ACCEPTED
[NO] DATA OVERRUN
[NO] DATA CHECK
[NO] FORMATTER PROM ERROR
[NO] FORMATTER CORRECTABLE ERROR
[NO] FORMATTER DATA BUS ERROR
```

MBTC exec level density: *ddddd (nn)*
This messages gives the name and number of the density setting.

MBTC exec level density: expected *ddddd (nn)*, **actual** *ddddd (nn)*
This message gives the names and numbers of the expected and actual density.

MBTC dstat[2] *0xnn* means:
{error message detail}

This gives the interpretation of the drive status byte at offset 0x22. (The interpretations appear in the next entry.)

MBTC dstat[2]: expected *0xnn*, actual *0xnn*, differences:
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x22. The message detail may be:

```
DIAG MODE LATCH [NOT] SET
[NO] UCE CONDITION
[NO] WRITE TAPE MARK CHECK
[NO] END DATA CHECK
[NOT] MTE CONDITION
[NO] VEL ERROR
```

MBTC dstat[3] *0xnn* means:
{error message detail}

This gives the interpretation of the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

MBTC dstat[3]: expected *0xnn*, actual *0xnn*, differences:
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

MBTC dstat[6] *0xnn* means:
{error message detail}

This gives the interpretation of the drive status byte at offset 0x26. (The interpretations appear in the next entry.)

MBTC dstat[6]: expected *0xnn*, actual *0xnn*, differences:
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x26. The interpretation may be:

```
DRIVE [NOT] READY
DRIVE [NOT] ONLINE
FILE [NOT] PROTECTED
[NOT] AT BEGINNING OF TAPE
[NOT] BACKWARD MOTION
[NOT] HIGH DENSITY
[NOT] PAST END OF TAPE
```

MBTC dstat[7] *0xnn* means:
{error message detail}
This gives the interpretation of the drive status byte at offset 0x27. The message may be:

```
[NOT] WRITE FUNCTION
```

MBTC dstat[7]: expected *0xnn*, actual *0xnn*, differences:
{error message detail}
This indicates a discrepancy in the drive status byte at offset 0x27. The message detail may be:

```
[NOT] WRITE FUNCTION
```

MBTC status *0xnn* means:
{error message detail}
This gives the interpretation of the controller status byte at offset 0x10. The message detail appears in the next entry.

MBTC status: expected *0xnn*, actual *0xnn*, differences:
{error message detail}
This indicates a discrepancy in the controller status byte at offset 0x10. The message detail may be:

```
BLOCK MODE
BUFFER A ACTIVE
BUFFER B ACTIVE
CHAIN MODE
COMMAND [NOT] EXECUTING
COMMAND [NOT] PENDING
DIAGNOSTICS [NOT] ENABLED
FORCE PARITY ERROR
INTERRUPT [NOT] ENABLED
[NO] INTERRUPT REQUEST
USE NORMAL PARITY
```

Pattern error at offset *0xnnnnnn* in buffer: expected *0xnn* actual *oxnn [0xnn 0xnn ...]*
This message indicates an error in data has been detected. The context following the erroneous byte may also be printed.

Retry count for this record: *nn*
This message gives the number of retries required to overcome a media error. This message is generally appended to other messages.

Tape block size error: Expected size: *nn*, actual size: *nn*
This message indicates an erroneous tape block size.

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

A

Alaska, reporting problems from, telephone number for x
Associated documents, how to order x
Associated documents, listed ix

C

C Programming Language ix
Canada, reporting problems from, telephone number for x
cattypedevnn.suffix 1-1
Cautions, described ix
Class 1 tests, controller loopback tests 4-8
Class 2 tests, operator controls tests 4-9
Command scripts, user-created 3-1
Command status, chaining commands 4-9
Command status, executing and pending commands 4-9
contact, aborting the report A-3, A-6
contact, editing the report A-6
contact, ending a response A-3
contact, ending the report A-6
.contact file, skipping first prompt by using A-3
contact, including files in your report A-5
contact, invoking A-1, A-4
contact, prerequisites A-1
contact, prompts A-4
contact, prompts, step-by-step discussion of A-4
contact, report, suspending A-3
contact, reporting problems A-1
contact, restrictions, on tilde-escape sequences A-5
contact, reviewing the report A-6
contact, skipping first prompt by using a *.contact* file A-3
contact, submitting *dead.report* file A-3
contact, submitting the report A-6
contact, tilde-escape sequences A-4
contact, tips on using A-2
Controller functional tests 4-10
Controller interrupt loopback 4-9
Controller reset 4-8
CONVEX, address, for ordering documents x
CONVEX Diagnostic Utilities Manual, C120 ix
CONVEX Diagnostic Utilities Manual, (C200 Series) ix
CONVEX Processor Operation Guide ix
CONVEX UNIX Tutorial Papers ix
CPU 1-1
CPU, *cpu*, test program for 1-2
cpu, test category 1-2

D

dead.report file, submitting A-3
dead.report file, using *-r* option to submit A-3
dev, test category 1-2
dev4200 (tape unit test) 4-1
dev4200 (test parameter menu) 4-3
Devices, *dev* for 1-1
Devices, test programs for, table 1-3
Devices, types, listed 1-2
Diagnostic environment, overview 1-1
Diagnostic shell. *See dshell*
Diagnostics, selecting 3-1
Disks 1-2
Disks, device, test program for 1-3
DMA FIFO board 4-9
DMA to entire memory space 4-9
Drive unload operation 4-10
dshell, introduction 3-1
dshell, overview 3-1

E

Error messages 4-16
Error messages, selecting 3-1
error reporting A-1

F

File protect status test 4-10
Files, test outputs to 3-1

H

Hawaii, reporting problems from, telephone number for x
Help-for *dev4200* prompts 4-3

I

I/O, subsystem test, *io* for 1-2
I/O system, test program categories for 1-1
io, test category 1-2

K

Kernel, hardware tests 1-2
Kernel, hardware tests, program for 1-3

M

mem, test category 1-2
Memory, subsystem test, *mem* for 1-2
Memory system, test program name for 1-1

N

Networks 1-2
Networks, device, test program for 1-3
Notational conventions, discussed ix
Notes, described ix

O

Offline tests 1-2
Offline tests, functional, program for 1-3
Online status test 4-9
Online tests 1-2
Online tests, functional, program for 1-3
Overview, diagnostic environment 1-1
Overview, *dshell* 3-1

P

Peripheral devices, test program name for 1-1
Peripherals, *dev*, test program for 1-2
Printers 1-2
Printers, device, test program for 1-3
problems, reporting, overview A-1

R

Reader's Forum x
Reporting problems x
Revision sheet 3

S

Screens, test outputs to 3-1
Scripts, predefined 3-1
Self-tests 1-2
Self-tests, test program for 1-3
Service Processor Unit. *See* SPU
SP2, subsystem test, *spu* for 1-2

SP2, *.t* programs and 1-1
SP2, test program name for 1-1
SPU, *dshell* and, introduction 3-1
spu, test category 1-2
Standalone tests 1-2
Subsystems, *cat* for 1-1

T

.t 1-1
TAC, reporting problems to x
TAC (Technical Assistance Center), problems, reporting to A-1
Tape, block read function 4-13
Tape, block skip backward 4-14
Tape, block skip forward 4-13
Tape, end-of-tape sensing 4-14
Tape, erase gap test 4-14
Tape, file skip backward 4-12
Tape, file skip forward 4-11
Tape, force parity error 4-14
Tape, read backward 4-13
Tape, read fixed length records 4-15
Tape, read forward 4-13
Tape, read variable length records 4-15
Tape subsystem class descriptions 4-7
Tape unit test 4-1
Tape units 1-2
Tape units, test program for 1-3
Tape, write 4-13
Tape, write chained data files 4-15
Tape, write fixed length records 4-14
Tape, write variable length records 4-15
Technical Assistance Center (TAC), problems, reporting to A-1
Technical assistance, discussed x
Terminals 1-2
Terminals, test program for 1-3
Test parameter menu 4-3
Test programs, categories 1-1
Test programs, categories, table 1-2
Test programs, device types 1-2
Test programs, naming conventions 1-1
Test programs, types 1-2
Test programs, types, table 1-2, 1-3
Test, read chained file 4-16
Tests, options, selecting 3-1
Tests, output, selecting 3-1
tilde-escape sequences A-4
tilde-escape sequences, restrictions on use A-5
Trouble reports x
trouble reports A-1

U

UNIX-to-UNIX Communication Protocol A-1
UNIX-to-UNIX copy command, *uucp* A-1
UUCP, connection to TAC A-1
uucp, UNIX-to-UNIX copy command A-1

V

vers, program version number found by using A-2

W

Warnings, described ix
whence, program path name found by using A-2
which, program path name found by using A-2
Write tape mark 4-11

CONVEX Multibus STC Tape Unit Controller
(*dev4200*) Diagnostics Manual
Document No. 760-002130-000
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)

CONVEX Multibus STC Tape Unit Controller
(*dev4200*) Diagnostics Manual
Document No. 760-002130-000
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)