northern
telecom

Problem Determination Tools User's Guide

Author(s):	Mikhail Khodosh
Manager:	Ian Hopper
Dept:	4Q21

Date:	April 9, 1993
Issue:	1.2
Project Name:	THOR
File Name:	pdt.ug.1.2.book
Activity Id:	DE0792
Release:	18

Keywords: PDT, THOR, Debugger, Problem Determination

Abstract: This document describes the THOR problem determination strategy.

Revision History

ISSUE NO.	DATE	AUTHOR	REASON FOR ISSUE
0.1	07/23/91	M.Khodosh	Initial release
0.2	01/17/92	M.Khodosh	Result from review
1.0	12/15/92	M.Khodosh	Gate II release
1.1	03/31/93	M.Khodosh	Gate III release
1.2	04/09/93	M.Khodosh	Gate III release (reviewed)

References

(1) Debug Tool Proposal, 10/21/88, Karl Bernhardt, Bob Asdel, Robert Stagmier, NT MTV

(2) Field Support Overlay Debug, User Manual, January/89, J. Pancevich

(3) Problem Determination Tools, High Level Design, 07/23/91, Mikhail Khodosh, NT MTV

(4) VxWorks Programmer's Guide, Wind River Systems, Inc., 1990

(5) Debug Notes (unpublished), Michael McKinney, NT MTV, 1993

Table of Contents

Table of Contents

Table of Contents

1 Introduction

1.1 Overview

Problem Determination Tools (PDT) is a collection of software tools which are intended to locate, examine and eliminate problems (malfunctions) in the Thor system.

In addition to its major function, PDT also performs two very important roles in the Thor environment:

- serves as a "secured gate" to Thor system;
- serves as a Thor system shell.

1.2 PDT Terminals

The user communicates with PDT via PDT terminals.

Any TTY connected to a CP serial port or to a SDI port can be used as a PDT terminal.

PDT also allows a remote user to communicate with PDT via Ethernet or over SLIP serial connection. In such a case, pseudo TTY device (PTY) is used as a PDT terminal instead of using a real TTY device. PTY simulates the serial I/O functions and looks like a real device for the PDT.

1.3 Implementation Notes

PDT is implemented as a set of related tasks running under the Thor Operating System. The most important PDT tasks are:

- pdtLogin - login and security features;
- pdtBrkTask - breakpoint handler;
- pdtShell*nn* - PDT shells (one per user).

2 Thor Operating System

2.1 Overview

The Thor Operating System is a multitasking real-time operating environment built upon the VxWorks OS from Wind River Systems Inc. It has been enhanced with several Thor-specific system tasks and creates a powerful environment in which SL-1 software runs as the main task. Most notable features are an exception/ recovery mechanism, MS-DOS compatible local file system and extensive UNIX-compatible networking facilities.

Because of its multitasking nature, Thor OS creates the appearance of many programs executing concurrently. Each apparently independent program is called a *task*.

2.2 Task Context

Each task has its own *context* which is CPU environment and system resources the task sees each time it is scheduled to run by the OS kernel. A task's context includes:

- a thread of execution, i.e., the task's program counter;
- CPU registers;
- a stack for dynamic variables and function calls;
- I/O assignments for standard input, output, and error.

One important resource that is not part of the task's context is memory address space. In Thor OS, all code executes in a single common address space.

2.3 Task State

For scheduling purposes, the Thor OS kernel maintains the current *state* of each task in the system. At each particular time each task can be in one of the following states:

ready	- the state of a task that is not waiting for any resource other than CPU;
pended	- the state of a task that is blocked due to the unavailability of some resource;
delayed	- the state of a task that is asleep for certain time period;
suspended	- this state inhibits task execution and is used primarily for debugging.

2.4 Task Name, ID and TCB

Each task in Thor OS has a unique name (an ASCII string of arbitrary length). In addition, when task is created, the system returns a *task ID* which is a four-byte pointer to the task's data structures. To reference a task, either task name or the task ID can be used.

Being a pointer, the task ID directly points to the most important task related data structure - Task Control Block (TCB). TCB contains or references all task-related data such as:

- task name and task ID;
- task status;
- stack base, stack size, current stack pointer, etc.;
- task registers.

2.5 "Current" Task

Many task control and debugging commands have a task parameter which is optional. If omitted, the *current* task is used. The current task is set when:

- A task hits a breakpoint or an exception.
- A task is single-stepped.
- Any command with a task parameter is executed with the task parameter specified.

2.6 Device List

Here is the list of standard device names used in Thor OS:

/null	- empty device;	
/lcd	- LCD display;	
/sio/0	- CP serial port 0;	
/sio/1	- CP serial port 1;	
/id0	- ID partition on hard disk 0;	
/id1	- ID partition on hard disk 1;	
/p	- protected hard disk partition;	
/u	- unprotected hard disk partition;	
/f0	- floppy disk 0 block device;	
/f1	- floppy disk 1 block device;	
/rf0	- floppy disk 0 raw device;	
/rf1	- floppy disk 1 raw device;	
/cart	- security card device;	
/pty/ptty0n.S	- pseudo tty slave device	(n = 0-7)
/pty/ptty0n.M	- pseudo tty master device	(n = 0-7)

3 PDT Operation

3.1 PDT Start-up

All PDT related tasks (except PDT Shell tasks) are created at the system start time. After proper initialization, PDT broadcasts the following message to both CP serial ports:

PDT Ready!

and is ready to operate. After that message, it is possible to start a PDT shell and to execute PDT commands.

PDT can be invoked by typing **^P^D^T**, or, for a remote user, by means of the **rlogin** command.

^P^D^T can also be used to restart the existing PDT shell.

Note. In earlier versions of release 18, **^P** is to be used instead of **^P^D^T**.

3.2 PDT Shell

The user interacts with PDT by means of a PDT shell. A PDT shell is created when **^P^D^T** or **rlogin** is entered. There can be several PDT shell tasks in the system at the same time depending on how many users interact with PDT simultaneously (one shell per user).

PDT shell is an interactive command interpreter. Although PDT has its own set of commands, it also allows to invoke any operating system subroutine. A PDT shell also provides the following mechanisms:

- standard input and output redirection;
- script execution;
- history facility.

3.3 Security levels

PDT security is provided by means of two mechanisms: access restriction (passwords) and access levels (security levels).

A password is required whenever a new PDT shell is created (as a result of a **^P^D^T** or **rlogin** command).

An access level is established separately for each PDT shell. A proper access level is selected automatically depending on the password provided by the user when PDT shell is created.

There are two security levels in PDT: *technical assistance* level and *system support* level.

On the technical assistance level, the restricted set of PDT commands is available.

On the system support level, the following features are available:

- full set of PDT commands;
- debug operation mode;
- superuser operation mode;
- direct access to operating system subroutines and data.

3.4 Operation Mode

There can be multiple PDT shells in the system at the same time. Although most of the PDT features can be accessed by multiple users simultaneously without any restrictions, there are some features which are restricted to use by only one user at a time. PDT has two special operation modes: *debug mode* and *superuser mode* which serve as mutual exclusion mechanisms to prevent multiple users from accessing these features at the same time. If any PDT shell switches to debug or superuser mode, no other PDT shell can switch to this mode until the first shell leaves this mode.

The following commands are available in the debug mode only: **b**, **br**, **bs**, **c**, **cret**, **s**, **so**, **freeze**, **unfreeze**, **ds** and **prs**.

In the superuser mode, PDT invokes VxWorks native shell and operates in its context. All VxWorks shell functionality is available in this mode.

If PDT shell is not switched to debug or superuser mode it operates in *regular mode*.

4 PDT Commands Summary

4.1 Debugging Facilities

4.1.1 Overview

Debugging facilities are one of the major PDT functionalities. PDT provides tools for debugging other tasks and SL1 task in the first place.

All PDT debugging activities are based on the *debugging event* and *debugging action* concepts.

Debugging event is a certain event, which occurrence causes the debugged task to be suspended. The only events that can be detected by PDT at this time are breakpoints. Breakpoints are created by means of **b** or **bs** commands.

Debugging action is a sequence of PDT commands executed for debugging purposes at the time when the debugged task is being suspended due to a debugging event. In general, the debugging process is iterative execution of the following three steps:

1. Interrupt the debugging task execution on a specified *debugging event*.
2. Perform required *debugging action*s, such as examine and change memory locations, stack and so on.
3. Resume the task execution (**c** command).

Debugging is mainly performed in the *interactive mode*. It means that when the task is suspended due to a *debugging event*, PDT will display the event description and then allow you to take any *debugging actions* by typing PDT commands one after another from the terminal.

To make the debugging process more efficient (less time-consuming), one can use PDT breakpoint macros.

4.1.2 Debug Mode

The most powerful (and most dangerous) PDT debugging facilities are available only in debug mode. Breakpoint-related commands, secondary memory freeze and unfreeze commands, secondary memory display commands are available only in debug mode. All breakpoint-related macros are executed in the debug mode as well.

The **debug** command is used to switch a PDT shell to the debug mode and switch it back to the regular mode.

Once a PDT shell is switched to debug mode, its terminal becomes the debugger terminal. The breakpoint hit indication messages will be sent to this terminal.

If you are in the debugging mode and need to use this terminal to communicate with SL-1, do NOT exit PDT shell. Use the **sl1input** command to switch terminal to SL-1 and **^P^D^T** sequence to switch terminal back to PDT.

4.2 Breakpoint Macro

Any PDT script can be used as a breakpoint macro. A macro can be bound to a breakpoint by means of a **bm** command. When a breakpoint is hit, the macro will be executed automatically before PDT switches to interactive mode. To prevent task suspension and switching to interactive mode due to breakpoint hit, include a **c** command into a breakpoint macro.

The following examples show how to use breakpoint macro. The session scenario is:

1. Create macro **ti.mac** using the **cat** command. This macro will print current task information and some virtual memory locations, and then continue execution.
2. Set breakpoint at **DIGPROC** in SL-1 code and bind **ti.mac** macro to this breakpoint.
3. To cause a breakpoint hit, dial a number on the phone set connected to the Thor machine.
4. Delete the breakpoint at **DIGPROC**.

macro creation example

```
pdt> cd /u/pdt
pdt> cat > ti.mac
ti
prp 8 6
c
^D
pdt> debug
pdt> b DIGPROC
0x43f58b8 (bp# 1): _DIGPROC          Task:       all   Count:  0
pdt> bm 1 ti.mac
pdt>
```

macro execution example

```
pdt>
Break at 0x43f58b8 (bp# 1): _DIGPROC        Task: 0x47d0000 (tSL1)
ti
    NAME        ENTRY       TID    PRI   STATUS      PC       SP
---------- ------------ -------- --- ---------- -------- --------
tSL1       _sl1Main      47d0000 240 SUSPEND     43f58b8  47d7ec8
stack: base 47d8000 end 47d0158 size 31956 high 5472  margin 26484
options: 0x10
VX_STDIO
D0 =       0  D4 = 2bae22  A0 =   20498  A4 =       0
D1 =       3  D5 =      0  A1 = 4b90058  A5 = 4b90000  SR =    3014
D2 = 2a1824  D6 =       0  A2 = 47d7ef4  A6 = 47d7ef4  PC = 43f58b8
D3 =       1  D7 =      0  A3 = 47d7f04  A7 = 47d7ec8
prp 8 6
00000008: 00000000 002c3be0  00007ede 00000207  *.....,;‘..~^....*
0000000c: 00000000 00000000                     *........        *
c
Break at 0x43f58b8 (bp# 1): _DIGPROC        Task: 0x47d0000 (tSL1)
ti
    NAME        ENTRY       TID    PRI   STATUS      PC       SP
---------- ------------ -------- --- ---------- -------- --------
tSL1       _sl1Main      47d0000 240 SUSPEND     43f58b8  47d7ec8
stack: base 47d8000 end 47d0158 size 31956 high 5472  margin 26484
options: 0x10
VX_STDIO
D0 =       0  D4 = 2bae22  A0 =   20498  A4 =       0
D1 =       3  D5 =      0  A1 = 4b90058  A5 = 4b90000  SR =    3014
D2 = 2a1824  D6 =       0  A2 = 47d7ef4  A6 = 47d7ef4  PC = 43f58b8
D3 =       1  D7 =      0  A3 = 47d7f04  A7 = 47d7ec8
prp 8 6
00000008: 00000000 002c3be8  00007ede 00000207  *.....,;‘..~^....*
0000000c: 00000000 00000000                     *........        *
c
Break at 0x43f58b8 (bp# 1): _DIGPROC        Task: 0x47d0000 (tSL1)
ti
    NAME        ENTRY       TID    PRI   STATUS      PC       SP
---------- ------------ -------- --- ---------- -------- --------
tSL1       _sl1Main      47d0000 240 SUSPEND     43f58b8  47d7ec8
stack: base 47d8000 end 47d0158 size 31956 high 5472  margin 26484
options: 0x10
VX_STDIO
D0 =       0  D4 = 2bae22  A0 =   20498  A4 =       0
D1 =       3  D5 =      0  A1 = 4b90058  A5 = 4b90000  SR =    3014
D2 = 2a1824  D6 =       0  A2 = 47d7ef4  A6 = 47d7ef4  PC = 43f58b8
D3 =       1  D7 =      0  A3 = 47d7f04  A7 = 47d7ec8
prp 8 6
00000008: 00000000 002c3c08  00007ede 00000207  *.....,;‘..~^....*
0000000c: 00000000 00000000                     *........        *
c
pdt> bd DIGPROC
pdt>
```

4.3 Secondary Memory Freeze and Display

Two PDT commands are intended to print secondary memory: **ds** prints real memory, and **prs** prints virtual memory. Both commands are available only in the debug mode and can operate only if the secondary memory has been frozen.

To freeze secondary memory, use the f**reeze** command. To unfreeze the secondary memory and synchronize both memories, use the **unfreeze** command. The memories synchronization is a time-consuming process. Please, BE PATIENT.

The **freeze** and **unfreeze** commands are available only in debug mode.

The following examples show how to freeze and display secondary memory. The session scenario is:

1. Use the **mstat** command to check memory status. Freeze secondary memory by means of the **freeze** command and then use **mstat** again to see the result.
2. For real memory print the same region in the primary and secondary memory using the **d** and **ds** commands, respectively (see the differences).
3. For virtual memory print the same region in the primary and secondary memory using **prp** and **prs** commands, respectively (see the differences).
4. Unfreeze the secondary memory by means of **unfreeze** command and check the final memory status.

freeze example

```
pdt> mstat
System mode:      Redundant.
   Side    Bank    Address range    Status   Read   Write
--------- ------ ----------------- -------- ------ -------
primary      1   4000000 - 43fffff   enbl    enbl   enbl
primary      2   4400000 - 47fffff   enbl    enbl   enbl
primary      3   4800000 - 4bfffff   enbl    enbl   enbl
primary      4   4c00000 - 4ffffff   enbl    enbl   enbl
primary      5   5000000 - 53fffff   enbl    enbl   enbl
primary      6   5400000 - 57fffff   enbl    enbl   enbl
primary      7   not populated
primary      8   not populated
secondary    1   4000000 - 43fffff   enbl    enbl   enbl
secondary    2   4400000 - 47fffff   enbl    dsbl   dsbl
secondary    3   4800000 - 4bfffff   enbl    enbl   enbl
secondary    4   4c00000 - 4ffffff   enbl    enbl   enbl
secondary    5   5000000 - 53fffff   enbl    enbl   enbl
secondary    6   5400000 - 57fffff   enbl    enbl   enbl
secondary    7   not populated
secondary    8   not populated
pdt> debug
pdt> freeze
Secondary memory frozen.
System is in non-redundant mode.
pdt> mstat
System mode:      Non-redundant.
   Side    Bank    Address range    Status   Read   Write
--------- ------ ----------------- -------- ------ -------
primary      1   4000000 - 43fffff   enbl    enbl   enbl
primary      2   4400000 - 47fffff   enbl    enbl   enbl
primary      3   4800000 - 4bfffff   enbl    enbl   enbl
primary      4   4c00000 - 4ffffff   enbl    enbl   enbl
primary      5   5000000 - 53fffff   enbl    enbl   enbl
primary      6   5400000 - 57fffff   enbl    enbl   enbl
primary      7   not populated
primary      8   not populated
pdt>
```

ds example

```
pdt> d 0x4717f7c 11 4
04717f70:                              00000000  *             ....*
04717f80: 00000000 00000006  04717f9c 04717f9c  *.........q...q..*
04717f90: 04717f9c 04062374  3000046d 7f980130  *.q....#t0..m...0*
04717fa0: 00000001 04717fb4                     *.....q.4        *

pdt> ds 0x4717f7c 11 4
04717f70:                              00000000  *             ....*
04717f80: 04717fa8 04717fc0  04717f9c 04624974  *.q.(.q.@.q...bIt*
04717f90: 04624960 00000000  04717fa8 04717fa8  *.bI‘.....q.(.q.(*
04717fa0: 046247ee 04717fb4                     *.bGn.q.4        *
pdt>
```

prs example

```
pdt> prp ?
Usage: prp [addr [,amount]]

pdt> prp 8 11
00000008: 00000000 00000000  00007ede 00000000  *..........~^....*
0000000c: 00000000 00000000  00000416 002f6688  *............./f.*
00000010: 0001f2f1 002f6788  0001f263           *..rq./g...rc    *

pdt> prs 8 11
00000008: 00000000 00000000  00007ede 00000000  *..........~^....*
0000000c: 00000000 00000000  00000000 00000000  *................*
00000010: 00000000 00000000  00000000           *............    *
pdt>
```

unfreeze example

```
pdt> unfreeze
Secondary memory unfrozen.
System is in redundant mode.
pdt> mstat
System mode:      Redundant.
   Side    Bank    Address range    Status   Read   Write
--------- ------ ----------------- -------- ------ -------
primary      1   4000000 - 43fffff   enbl    enbl   enbl
primary      2   4400000 - 47fffff   enbl    enbl   enbl
primary      3   4800000 - 4bfffff   enbl    enbl   enbl
primary      4   4c00000 - 4ffffff   enbl    enbl   enbl
primary      5   5000000 - 53fffff   enbl    enbl   enbl
primary      6   5400000 - 57fffff   enbl    enbl   enbl
primary      7   not populated
primary      8   not populated

secondary    1   4000000 - 43fffff   enbl    enbl   enbl
secondary    2   4400000 - 47fffff   enbl    dsbl   dsbl
secondary    3   4800000 - 4bfffff   enbl    enbl   enbl
secondary    4   4c00000 - 4ffffff   enbl    enbl   enbl
secondary    5   5000000 - 53fffff   enbl    enbl   enbl
secondary    6   5400000 - 57fffff   enbl    enbl   enbl
secondary    7   not populated
secondary    8   not populated
pdt> debug end
pdt>
```

4.4 Monitoring SL1 Activity

SL1 Spy facilities allow to collect and print SL1 work-load data. Data collection is performed periodically with a user-defined frequency. Each periodical data snap (sample) includes the following SL1 data:

- current and average number of interrupts for each group;
- current and average value of call processing Work Counts;
- current and average number of entries in call processing queues.

In addition to SL1 data, each sample includes:

- side where data has been collected;
- sample number and time stamp.

Two commands control SL1 monitoring: **sl1Spy** starts data collection and printing, and **sl1SpyStop** terminates the process. Two other commands provide collecting data report facilities: **sl1SpyTail** allows to print data samples that have already been collected and **sl1SpyWatch** sets the mode to print upcoming data samples. **sl1SpyHide** allows to suspend printing of data started by **sl1Spy** or **sl1SpyWatch**.

The following examples illustrate how SL1 Spy commands are used. The session scenario is:

1. Activate SL1 Spy with a 30 second period by means of **sl1Spy** command and stop printing after two samples by means of **sl1SpyHide**.
2. Resume printing by means of **sl1SpyWatch** and stop it after three samples by means of **sl1SpyHide**.
3. Print last two samples by means of **sl1SpyTail**.
4. Stop SL1 Spy by means of **sl1SpyStop**.

sl1Spy example

```
pdt> sl1Spy 30
   Starting sl1Spy. Sampling every 30 secs.
   To stop: use 'sl1SpyStop'
   To hide: use 'sl1SpyHide'
task spawned: id = 0x4186d98, name = t2
value = 0 = 0x0
pdt>
SPY 0: sample 1 (16/3/93 17:14:32)
  Interrupts        READY             IO           LINT
          0:       1133(  1133)     0(     0)     17(    17)
          4:        117(   117)     0(     0)     48(    48)
  WorkCounts
       Wdog:     209749(     209749)
       Idle:   25931040(   25931040)
  WorkQueues
    Cadence:          3(   queue  2)
       2Sec:          1(   queue  4)
       Idle:        193(   queue 12)
SPY 0: sample 2 (16/3/93 17:15:03)
  Interrupts        READY             IO           LINT
          0:        318(   500)     0(     0)      2(    12)
          4:          3(    48)     0(     0)      0(    16)
  WorkCounts
       Wdog:     214911(     213771)
       Idle:   26852160(   26545080)
  WorkQueues
    Cadence:          2(   queue  2)
       Dial:          1(   queue  6)
       Idle:        193(   queue 12)
 sl1SpyHide
 value = 0 = 0x0
 pdt>
```

sl1SpyWatch example

```
pdt> sl1SpyWatch
   The most current SL1 SPY Record:
SPY 0: sample 3 (16/3/93 17:15:32)
  Interrupts        READY             IO            LINT
          0:          49(   591)     0(      0)     18(     17)
          4:          24(    70)     0(      0)      0(     24)
  WorkCounts
       Wdog:      216654(     213201)
       Idle:    26852160(   26391600)
  WorkQueues
    Cadence:           3(   queue  2)
       Idle:         194(   queue 12)
value = 0 = 0x0
pdt>
SPY 0: sample 4 (16/3/93 17:16:01)
  Interrupts        READY             IO            LINT
          0:         318(   500)     0(      0)      2(     12)
          4:           3(    48)     0(      0)      0(     16)
  WorkCounts
       Wdog:      214911(     213771)
       Idle:    26852160(   26545080)
  WorkQueues
    Cadence:           2(   queue  2)
       Dial:           1(   queue  6)
       Idle:         193(   queue 12)
SPY 0: sample 5 (16/3/93 17:16:29)
  Interrupts        READY             IO            LINT
          0:        1804(   826)     0(      0)     25(     15)
          4:          83(    56)     0(      0)     49(     24)
  WorkCounts
       Wdog:      206146(     211865)
       Idle:    26642880(   26569560)
  WorkQueues
    Cadence:           2(   queue  2)
    128LowP:           1(   queue  3)
       Dial:           1(   queue  6)
       Idle:         193(   queue 12)
sl1SpyHide
value = 0 = 0x0
pdt>
```

sl1SpyTail example

```
pdt> sl1SpyTail 2
SPY 0: sample 10 (16/3/93 17:19:22)
  Interrupts        READY              IO            LINT
          0:        302(   583)    0(     6)      0(    13)
          4:          3(    62)    0(     0)      0(     9)
  WorkCounts
       Wdog:     213014(     213095)
       Idle:   27104280(   26512440)
  WorkQueues
       Idle:        197(   queue 12)
SPY 0: sample 11 (16/3/93 17:19:51)
  Interrupts        READY              IO            LINT
          0:       1803(   694)    0(     5)      8(    12)
          4:         83(    64)    0(     0)      0(     9)
  WorkCounts
       Wdog:     208975(     212721)
       Idle:   26514720(   26512680)
  WorkQueues
       Idle:        197(   queue 12)
value = 0 = 0x0
pdt>
```

sl1SpyStop example

```
pdt> sl1SpyStop
... sl1Spy Data Collection has been completed.
    Use 'sl1SpyTail N' to show latest N (up to 64) records
value = 0 = 0x0
pdt>
```

Multiple users can use SL1 SPY simultaneously. However, one user's action may affect others. For example, **sl1SpyWatch** output will be redirected to the latest terminal that issued this command.

An on-line SL1 Spy help is provided by means of the **sl1SpyHelp** command.

4.5 Report File

4.5.1 Overview

Thor reporting facilities maintain a special file to log reports produced by Thor subsystems. This report file has a constant size and is maintained on a circular basis (when the size is reached, a new report replaces the oldest one). By default, the report file is placed in the '**/u/rpt**' directory and has the name '**rpt.log**'.

The report file is stored in a binary format. It can be converted into an ASCII text file by means of a special utility program **rptLogConvert**.

In most cases, including debugging, it is not necessary to convert the entire report file into ASCII format. Instead, the report file browsing facility allows to look through the file and display relevant records only.

4.5.2 Report File Browsing

Report file browsing commands allow to look through the report file and display selected reports. It is possible to process the report file sequentially or go through the file in arbitrary order. To make sequential processing possible, the browsing facility keeps track of the last displayed report (*current report*).

None of the browsing commands, except **rdopen**, has a *fileName* parameter. It means that all these commands work on the current file being browsed. In most cases, the current file is a system report log file **/u/rpt/rpt.log**.

Two commands control the file being browsed: **rdopen** and **rdshow**. The **rdopen** command allows to specify explicitly the name of the file to browse on and is to be used if it is necessary to browse a file other the default report file. It is sometimes useful to save a copy of the **/u/rpt/rpt.log** file under different name to analyze reports later. The **rdshow** command shows the current file name and other attributes.

In the following example, we check the current file attributes, then open the **/u/rpt/rpt.log** file (by default), and then open a different report file that has been saved earlier under the **tt930318.lograw** name.

rdopen and **rdshow** examples

```
pdt> rdshow
... report file not selected. Use 'rdopen' or see 'rdhelp 1'
value = -1 = 0xFFFFFFFF
pdt> rdopen
           Work file :  "/u/rpt/rpt.log"
         File status :  partially-full
       File capacity :  738
          oldest rec :    0 (25/ 3/93 13:06:53)
         current rec :  657 (25/ 3/93 17:50:39)
          newest rec :  657 (25/ 3/93 17:50:39)
        display size :   16 (25/ 3/93 18:53:35)
value = 0 = 0x0
pdt> cd /tmp/oldrpt
pdt> rdopen tt930318.lograw
...rd : 556 new reports arrived since last command
           Work file :  "tt930318.lograw"
         File status :  partially-full
       File capacity :  738
          oldest rec :    0 (17/ 3/93 15:55:36)
         current rec :  474 (18/ 3/93 09:26:59)
          newest rec :  474 (18/ 3/93 09:26:59)
        display size :   16 (25/ 3/93 19:01:27)
value = 0 = 0x0
pdt>
```

The following two commands allow to perform sequential processing of the report file:

rd [*S*] [*R*]

rds [*S*] [*R*]

First, they skip *S* number of reports starting from the *current report* and then display *R* consecutive reports. Both parameters can be signed values. The positive value stands for forward skipping or displaying and the negative value stands for backward skipping or displaying.

The **rd** command displays a short report description, whereas the **rds** command provides more complete information, including task registers, return address stack and data stack printouts. The **rds** command creates symbolic output based on the information in the system symbol table. The symbol table should be loaded prior to **rds** command execution (see **symload** command).

In the following example, we use the **rd** command to skip 4 reports and display next 2 reports, and then use the **rds** command to skip 2 reports backward and display the same reports as in the previous **rd** command.

rd and **rds** examples

```
pdt> rd 4 2
  370 : BERR0704 EXC 0: Bus Error in ISR
          SR=0x2700, PC=0x46d7ed2, Addr=0x13030000, SSW=0x074d (25/3/93 10:18:34.639)
  371 : SRPT0782 RST 0: WARM START IN PROGRESS - Reason 42 (25/3/93 10:18:34.654)
value = 0 = 0x0
pdt> rds -2 2
... rd : record 370
BERR0704 EXC 0: Bus Error in ISR
        SR=0x2700, PC=0x46d7ed2, Addr=0x13030000, SSW=0x074d (25/3/93 10:18:34.639)
Registers (A0-A7, D0-D7):
 00100002 0569e604 0410e0a4 04717f9c  00000000 0410e064 0410dfd0 0410dd24
 00000000 00000000 00000000 0410e064 00000002 04072d9e 0000000f 00000000
Interrupt level: 1
Return Address Stack:
 04073054 (_exchThreshold+94)
 04072d54 (_giaExcISR+2a)
 04072c48 (_giaExcHandler+24)
 046d7e76 (_NEXT_ELEMENT+20)
 046d7dae (_DIVIDE2+2a)
 046d863a (_getTdb+7a)
 046d8b6e (_SET_HEX_DISPLAY+34)
 040437ce (_sysClkInt+1a)
 04072c76 ( _giaIntHandler+26)
 0462220e (__TTR_INPUT+cc0)
 046221a6 (__TTR_INPUT+c58)
 0462207c (__TTR_INPUT+b2e)
Stack (base = 0x410dfd8):
 0000074d  0410e064 0410e0a4   00000002  00c0c000  0569e670  0410e020  04064650
 0410e024  040b90aa  041f68c4   041f68dc   042db60c  0410e048  00000018  0410e054
 0410e050  04072d54  00000002  0410e0a4  0410e064  00c0c000  00000000  00000000
 0410e108  04072c48  00000002  0410e0a4  0410e064  00030000  00000000  00000000
 00c0c000  00000000  00000000  00c0c000   00000000 13000000  0569e604   0569e670
 04717f9c   00000000  04ab0000  0410e108  0410e0a0  2700046d  7ed2b008  1eee074d
 0800000d 13030000  13030000   00030000  20300800  046d7eda  046d7ed8  046d7ed6
 00000000  08000db1  000ff487    00000000  00000001  0000ffff 0   00000008  1207ba0
 13030000  00000000  0000c000  13030000  04ab0000  0569e604  0410e134  046d7e76
... rd : record 371
SRPT0782 RST 0: WARM START IN PROGRESS - Reason 42 (25/3/93 10:18:34.654)
Registers (A0-A7, D0-D7):
 00100002  057f9a04   00000000  04717f9c   00000000  0410e064  0410df94   0410dce8
 00000000  00000000  00000000  00000000  0000002a  00000002  00c0c000  00000000
Interrupt level: 1
Return Address Stack:
 0409b126 (_rstWarmStart+4e)
 0404374c (_sysToMonitor+44)
 040b29f8 (_rebootHookAdd+a)
 040d5816 (_excExcHandle+94)
 04072d68 (_giaExcISR+3e)
 04072c48 (_giaExcHandler+24)
 046d7e76 (_NEXT_ELEMENT+20)
 046d7dae (_DIVIDE2+2a)
 046d863a (_getTdb+7a)
 046d8b6e (_SET_HEX_DISPLAY+34)
. . . . . . . . . . . . . . . . . . . . . . .
```

The next two commands allow to go through the report file in arbitrary order:

rdgo[*repnum*]

rdtime[*time* [*date*]]

The **rdgo** command goes to the specified report and displays it. The **rdtime** command goes to the report at or before the specified time stamp and displays it. Both commands change the *current report* position in the file to the displayed one.

In the following example, we use the **rdshow** command to look at the current report number, then go and display report number 107 by means of the **rdgo** command, and then go and display the report produced at 7 p. m. on the 1st of April by means of the **rdtime** command. Pay attention that the time stamp in the displayed report is not exactly the same as it was specified in the **rdtime** command.

rdgo and **rdtime** examples

```
pdt> rdshow
            Work file :  "/u/rpt/rpt.log"
          File status :  full(old reports are replaced by new ones)
        File capacity :  738
           oldest rec :  105 (26/ 3/93 12:29:59)
          current rec :  104 (26/ 3/93 15:00:13)
           newest rec :  104 (26/ 3/93 15:00:13)
         display size :   10 (26/ 3/93 16:27:58)
value = 0 = 0x0
pdt> rdgo 387
  387 : BERR0705 EXC 1: Bus Error in Task "tSL1" (0x4710000)
        SR=0x3000, PC=0x46d7702, Addr=0x13020f10, SSW=0x074d (26/3/93 13:05:16.599)
value = 0 = 0x0
pdt> rdtime 13 05 20 03 26
... rdTime: searching for record at/before: ( 3/26/93 13:05:20)
... rdTime: going to the newest record (rec 104)
rec 104:
  104 : CCED0762 SWO 0: Graceful switch-over to side 0 completed
        Previous Graceful SWO: at 26/3/93 14:59:36 (26/3/93 15:00:13.680)
  105 : HWI0003 HI Init: Graceful SWO Start continues on side 1 (26/3/93 12:29:59.019)
value = 0 = 0x0
pdt>
```

Two more commands are based on report time stamps:

rdhead[*N*]

rdtail[*N*]

The **rdhead** command displays *N* oldest (earliest in time) reports, and the **rdtail** command displays *N* newest (latest in time).

The **rdall** command displays all reports which are in the report log file.

An on-line help for the report log file browsing facility is provided by means of the **rdhelp** command.

Note: Reports without time-stamp (ROM reports) may confuse **rdtime** command.

4.6 Remote File Transfer

You can use a FTP protocol to transfer files between the THOR machine and the UNIX workstation over serial lines. For this purpose, it is necessary to make some coordinated efforts on both ends of serial connection.

In the following scenario, it is assumed that the entire session takes place in the same workstation window.

To prepare for file transfers:

on the THOR end:

1. Start PDT shell on any CP serial port. If you are a remote PDT user, dial into PDT via modem.
2. Issue a **slipBegin** command to put the THOR serial port into SLIP mode. PDT will prompt a command to be entered on the UNIX end:

   "slipBegin <tty><baudrate>".
3. Drop the connection by means of "**~.**" or "**~^D**".

on the UNIX end:

1. Issue "**slipBegin** <tty> <baudrate>" command to put UNIX <tty> into SLIP mode.
2. Start the FTP program using the "**ftp thor**" command.

Now you can use FTP commands to do required file manipulations.

When file transfers are all completed:

on the THOR end:

1. Start PDT shell using "**rlogin thor**" command.
2. Issue **slipEnd** command to PDT. This should be followed by an extra <cr> (it is very important to enter two <cr> at that point).

on the UNIX end:

1. Issue **slipEnd** command to UNIX.
2. Regain access to PDT using "**tip** <tty>" command.

During a file transfer, you can actually use PDT from another window (using another "rlogin"), but it can be very slow. Nevertheless, it is often useful to use another window to determine which files to transfer and ensure that the file lengths compare.

4.7 Task Control

Most of the PDT task control commands are used to provide various information about a specific task or about all tasks in the system. As a rule, all these commands allow to reference a task by either task name or task ID.

The following task control commands are available in PDT:

i [*task*]	- Information. This command gives a snapshot of the tasks currently running in the system and some information about each of them, such as task name, task ID (TCB address), task state, program counter, stack pointer, etc.
ti [*task*]	- Task information. This command gives complete information about a specific task contained in the task TCB. It includes everything shown for that task by **i** command plus all task registers, additional information about stack usage, etc.
tt [*task*]	- Task trace. This command traces a task’s return address stack, shows what routine the task is currently executing and how it got there.
td *task*	- This command makes the specified task exit and deallocates the stack.

If *task* parameter is omitted in **ti** or **tt** command, the current task is used.

4.8 On-line help

The **help** command provides a list of available PDT commands. The set of available commands depends on a security level established for a particular PDT shell.

In addition to this common help, most of the PDT commands have its own help option. Being invoked with the only operand, a question mark (**"?"**), each command will respond with the "usage" information.

Appendix A Command Reference

Introduction

This appendix is a command reference containing descriptions of PDT commands. The commands are listed in an alphabetical order. Each command description contains a command format, operands, functional description, operation mode, list of related commands, and an examples of command invocation. When a command does not have pertinent information for one of these categories, the category is omitted from command description. For example, the **pwd** command does not have any operands, so that paragraph is not included in the **pwd** command description.

The following typographical conventions are used in commands formats and descriptions:

pwd	- command name or parameter, that should be typed as shown;
addr	- user-supplied parameters;
[ ]	- indicate that this parameter is optional;
[*filename* ...]	- indicate that this parameter can be repeated several times;
index | **all**	**-** indicate a choice;
^**C**	- control characters;
<sp>	- space;
<cr>	- carriage return.

ADDRESS

COMMAND FORMAT:

adr *global offset*

OPERANDS:

global	- global number (hexadecimal).
offset	- byte offset (hexadecimal).

DESCRIPTION:

The **adr** command displays real and virtual addresses and the contents of the memory location corresponding to the specified parameters.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
adr 14a 41e

real addr:      0x000433b6aa
virtual addr:   00ffe22daa
op code:        000a4efb
```

SEE ALSO:

fgn - find global number.

BREAKPOINT SET

COMMAND FORMAT:

b [*addr*[, *task* [, *passcount*]]]

OPERANDS:

addr	- main memory address where to set a breakpoint.
task	- task for which to set a breakpoint.
passcount	- number of passes before hit.

DESCRIPTION:

The **b** command allows to set or display breakpoints.

To display the list of currently active breakpoints, call it without arguments. The list shows the address, task, and pass count of each breakpoint.

To set a breakpoint, include the *addr*, which can be specified numerically or symbolically with an optional offset.

If *task* is specified, the breakpoint will apply to a specific task. If *task* is omitted, the breakpoint will apply to all breakable tasks.

If *passcount* is zero omitted, the break will occur every time it is hit. If *passcount* is specified, the break will not occur until the passcount+1th time an eligible task hits the breakpoint, i.e., the breakpoint is ignored the first *passcount* times it is hit.

Individual tasks can be unbreakable, in which case breakpoints that otherwise would apply to a task are ignored.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLES:

`b`	- `list all existing breakpoints`
`b DIGPROC`	- `set a breakpoint for all tasks at the memory location corresponding to DIGPROC symbol`
`b 0x48c2900, tSL1`	- `set a breakpoint for tSL1 task at the 0x4024567 memory address`

SEE ALSO:

bd	- breakpoint delete.
bs	- breakpoint suspend.
debug	- turn debug mode on / off.

BREAKPOINT DELETE

COMMAND FORMAT:

bd *addr*[, *task*] | all

bdall

OPERANDS:

addr	- main memory address where to set a breakpoint.
task	- task for which to set a breakpoint.

DESCRIPTION:

These commands are intended for deleting breakpoints.

In the **bd** command if *task* is omitted, the breakpoint will be removed for all tasks. If the breakpoint applies to all tasks, removing it for only one task will be ineffective. It must be removed for all tasks and then set for just those tasks desired.

The **bdall** command as well as **bd** command with the 'all' parameter allows to delete all existing breakpoints.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLES:

```
bd TCM_INPUT_MSG     - delete a breakpoint for all tasks at
                       TCM_INPUT_MSG memory location
bd DIGPROC, tSL1     - delete breakpoint at DIGPROC for tSL1 task
                       only
bdall                - delete all breakpoints for all tasks
```

SEE ALSO:

b	- breakpoint set.
bs	- breakpoint suspend.
debug	- turn debug mode on / off.

BREAKPOINT MACRO

COMMAND FORMAT:

bm *brkptno* [, *macro*]

OPERANDS:

brkptno	- breakpoint number.
macro	- disk file which contains predefined PDT macro script.

DESCRIPTION:

The **bm** command binds the breakpoint specified by *brkptno* with the *macro* (script consists of PDT commands) to be executed upon encountering this breakpoint.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

```
bm 2, /f0/mac1.cmd   - bind breakpoint number 2 with the script file
                       mac1.cmd located on the /f0 device (floppy
                       drive)
```

SEE ALSO:

b	- breakpoint set.
bl	- breakpoint list.
bs	- breakpoint suspend.
debug	- turn debug mode on / off.

BREAKPOINT RESUME

COMMAND FORMAT:

br [*addr*[, *task*]]

OPERANDS:

addr	- main memory address of the existing breakpoint.
task	- task for which the breakpoint was set.

DESCRIPTION:

The **br** command allows to resume a breakpoint which was suspended by the **bs** command. The breakpoint becomes active and fully operative as it was before it had been suspended.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLES:

`br aFunc`	- resume a breakpoint at the memory location corresponding to 'aFunc' symbol
`bd aFunc, aTask`	- resume a breakpoint which was set for 'aTask' task at the memory location corresponding to 'aFunc' symbol

SEE ALSO:

b	- breakpoint set.
bs	- breakpoint suspend.
debug	- turn debug mode on / off.

BREAKPOINT SUSPEND

COMMAND FORMAT:

bs *addr*[, *task* [, *passcount*]]

OPERANDS:

addr	- main memory address where to set a breakpoint.
task	- task for which to set a breakpoint.
passcount	- number of passes before hit.

DESCRIPTION:

The **bs** command allows to suspend (disable) the existing breakpoint or set a new breakpoint and immediately suspend it. Rather than deleting a breakpoint, it may be more appropriate to suspend it. The **bs** command makes the breakpoint inoperative as if it had been deleted, but remembers the information about the breakpoint so that it can be resumed (enabled) again later.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

```
bs REQUEST_OUTPUT    - suspend a breakpoint  for  all  tasks  at
                       REQUEST_OUTPUT memory location
```

SEE ALSO:

b	- breakpoint set.
br	- breakpoint resume.
debug	- turn debug mode on / off.

CONTINUE

COMMAND FORMAT:

c [*task*[, *addr*]]

OPERANDS:

task	- task to be continued.
addr	- main memory address to continue at.

DESCRIPTION:

The **c** command allows to continue execution from a breakpoint or single-step.

If *task* is omitted, the last task referenced is assumed. If *task* is not at a breakpoint, nothing happens.

If *addr* is omitted, the task continues execution, starting with the instruction where the breakpoint was located. If *addr* is specified, the program counter is changed to *addr*, and the task is continued.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLES:

`c`	- continue the last referenced task.
`c aTask`	- continue aTask task.

SEE ALSO:

b	- breakpoint set.
bs	- breakpoint suspend.
cret	- continue until return.
debug	- turn debug operation mode on / off.
s	- single-step a task.
so	- step over a subroutine.

CONCATENATE AND DISPLAY

COMMAND FORMAT:

cat [*filename1* [, *filename2*]...]

OPERANDS:

filename	- name of a file to be displayed.

DESCRIPTION:

The **cat** command reads each *filename* in sequence and displays it on the standard output device. If no *filename* argument is specified, **cat** reads from the standard input device. If the standard input device is a terminal, input is terminated by an EOF condition (CTRL-D).

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

`cat file1.txt`	- display contents of file1.txt file on the standard output device
`cat file1.txt, file2.txt > file3.txt`	- concatenate the first two files and place the result on the third.

SEE ALSO:

type - display the contents of the file on standard output device.

CHANGE DIRECTORY

COMMAND FORMAT:

cd [*directory*]

OPERANDS:

directory - pathname of the new working directory.

DESCRIPTION:

The **cd** command changes the current (working) directory to the specified *directory*.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
cd /u/log
```
- make /u/log the current directory.

SEE ALSO:

pwd - display the pathname of the working directory.

CHECK DISK

COMMAND FORMAT:

chkdsk [*device*]

OPERANDS:

device - disk device name.

DESCRIPTION:

The **chkdsk** command displays the volume label and memory status for the specified disk device.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

`chkdsk /u` - print memory status for the /u partition of the hard disk.

CLEAR MFC SIGNAL MONITOR

COMMAND FORMAT:

csm [*index* | **all**]

OPERANDS:

index - index in the array of the monitored TN(s).

DESCRIPTION:

The **csm** command clears one or all TN(s) being monitored during MFC signalling. If ALL is specified, all monitored TN(s) will be cleared. If *index* is specified, the o TN corresponding to this *index* in the array of monitored TNs will only be cleared.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
csm 1
TNM  01  0000

csm all
TNM  00  0000
TNM  01  0000
TNM  02  0000
TNM  03  0000
TNM  04  0000
TNM  05  0000
TNM  06  0000
TNM  07  0000
TNM  08  0000
TNM  09  0000
```

SEE ALSO:

dsm	- display TN(s) being monitored during MFC signalling.
ssm	- set MFC signal monitor.
tsm	- select type of MFC signal monitor message.

COPY FILES

COMMAND FORMAT:

cp *filename1*, *filename2*

cp *filename1*[, *filename2*...] *directory*

OPERANDS:

filename	- name of a file to be copied.
directory	- pathname of the destination directory.

DESCRIPTION:

The first form of a **cp** command copies contents of *filename1* onto *filename2.*

The second form, copies each *filename* to the specified *directory*; the "basename" of the copy corresponds to that of the original. The destination directory must already exist for the copy to succeed.

cp refuses to copy a file onto itself.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

ALIAS: **copy**

EXAMPLE:

`cp prog.abc, example`	- copy file 'prog.abc' from the current directory onto the file 'example' in the same directory.
`cp /p/prog.abc, /p/example .`	- copy files 'prog.abc' and 'example' from the '/p' directory to the current directory.

SEE ALSO:

mv	- move files.
ren	- rename file.

CONTINUE UNTIL RETURN

COMMAND FORMAT:

cret [*task*]

OPERANDS:

task	- task to be continued.

DESCRIPTION:

The **cret** command places a breakpoint at the return address of the current subroutine of a specified *task* and then continues execution of that task. When the breakpoint is hit, information about the task will be printed in the same format as in single-stepping. The breakpoint is automatically removed when hit, or if the task hits another breakpoint first. If *task* is omitted, the last task referenced is assumed.

This command can be useful for examining return values.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

`cret`	- continue the last referenced task until return from subroutine.

SEE ALSO:

b	- breakpoint set.
bs	- breakpoint suspend.
c	- continue task execution.
s	- single-step a task.
so	- step over a subroutine.

CALL REGISTER

COMMAND FORMAT:

crg *tn*

OPERANDS:

tn	- terminal number. Can be specified as a hexadecimal number or in the form of l s c u.

DESCRIPTION:

The **crg** command displays a snapshot of the call register for the specified *tn* at the time when the command is entered. This information includes the unit type, customer number, pointers to protected and unprotected line blocks, ACTIVECR (if any), heldcr (if any), and SL-1 set keylinks (if applicable). Attendant information is displayed by the active loop with the information being the source, destination and held CRPTR for the loop.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
crg 24 0 0 1

SLOOP TN  001801  EQPD
CUST  1 PBLK 0001E99B UBLK 002F42C4
BCS
```

SEE ALSO:

tnt	- TN translation.

DISPLAY MEMORY

COMMAND FORMAT:

d [*addr* [, *amount* [, *unit*]]]

OPERANDS:

addr	- memory address to start from.
amount	- amount of memory to be printed (in units defined by *unit*).
unit	- 1, 2, or 4. Memory will be read as 1, 2 or 4 bytes depending on the specified value.

DESCRIPTION:

The **d** command displays a specified *amount* of *units* starting from *addr*.

If *addr* is not specified, the location where the previous **d** command has terminated will be used. If *amount* or *unit* is not specified its value from the previous **d** command will be used.

Memory is displayed in two formats: hexadecimal and ASCII. In ASCII portion unprintable characters are indicated by a period(.).

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE

d program, 10, 4	- display 10 long words (4-byte-units) starting from the memory location corresponding to 'program' symbol.
d 0x040d3448, 10	- display memory starting from the location with the hexadecimal address 0x040d3448. Memory amount and unit size will be taken from the previous **d** command.

SEE ALSO:

ds	- display secondary memory.
p, prp, prs	- print virtual memory.

DISPLAY CUSTOMER POINTERS

COMMAND FORMAT:

dcp *customer*

OPERANDS:

customer	- customer number (decimal).

DESCRIPTION:

The **dcp** command displays PCDATAPTR and UCDATAPTR data pointers for a specified *customer*.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

To display data pointers for customer number 2, the following command should be entered:

```
dcp 2
```

As a result, the following information can be displayed:

```
CUST 2  P 01512A  U 2F9708
```

SEE ALSO:

dnt	- DN translation.
drp	- display route pointers.

DEBUG MODE

COMMAND FORMAT:

debug [**end**]

DESCRIPTION:

The **debug** command turns on and off the debug operation mode of the current PDT shell. There are some PDT command groups (most of them for debugging purposes) which can be used only by one PDT shell task at a time. The **debug** command provides a mutual exclusion mechanism to prevent multiple PDT shells from accessing these facilities at the same time. If the **debug** command has been entered for some PDT shell, no other PDT shell can switch to the debug mode until the first shell leaves debug mode in response to **debug end** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular, debug**

EXAMPLES:

```
debug       - turn debug mode on.
debug end   - turn debug mode off.
```

SEE ALSO:

b, **br**, **bs**, **c**, **cret**, **s**, **so**;

ds, **freeze**, **unfreeze**.

DEVICES

COMMAND FORMAT:

devs

DESCRIPTION:

The **devs** command prints a list of all devices which the I/O system knows about.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> devs
drv name
  0 /null
  1 /lcd
  2 /sio/0
  2 /sio/1
  4 /id0
  4 /id1
  4 /f0
  4 /f1
  4 /u
  4 /p
  5 /rf0
  5 /rf1
  6 /cart
  9 /pty/ptty00.S
 10 /pty/ptty00.M
  9 /pty/ptty01.S
 10 /pty/ptty01.M
  9 /pty/ptty02.S
 10 /pty/ptty02.M
 ................
```

DN TRANSLATION

COMMAND FORMAT:

dnt *customer dn*

OPERANDS:

customer	- customer number (decimal).
dn	- directory number (decimal).

DESCRIPTION:

The **dnt** command displays information about the *dn* directory number for the specified *customer*. The displayed information will include the unit type and either TN, DNBLOCK, or ROUTE DATA BLOCK. The route data block in turn will be displayed in three sections:

- Protected Route data block
- TRKLIST
- Unprotected Route data block

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
dnt 0 5016
DIG 1 INV
```

SEE ALSO:

dcp	- display customer pointers.
drp	- DN translation.

DISPLAY ROUTE POINTERS

COMMAND FORMAT:

drp *customer route*

OPERANDS:

customer	- customer number (decimal).
route	- route number (decimal).

DESCRIPTION:

The **drp** command displays customer route pointers in the following order: PRB_PRT, URB_PRT, TLIST_PRT, RRB_PRT.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

To display route pointers for customer number 2, the following command should be entered:

```
drp 0 12
CUST 0 EQUT 12 P 0001778E U 002F8B85 T 00028D40 R 00000000
```

SEE ALSO:

dcp	- display customer pointers.
dnt	- DN translation.

DISPLAY SECONDARY MEMORY

COMMAND FORMAT:

ds [*addr* [, *amount* [, *unit*]]]

OPERANDS:

addr	- secondary memory address to start from.
amount	- amount of memory to be printed (in units defined by *unit*).
unit	- 1, 2, or 4. Memory will be read as 1, 2 or 4 bytes depending on the specified value.

DESCRIPTION:

The **ds** command displays a specified *amount* of *units* starting from *addr* in secondary memory.

If *addr* is not specified, the location where the previous **ds** command has terminated will be used. If *amount* or *unit* is not specified its value will be taken from the previous **ds** command.

Memory is displayed in two formats: hexadecimal and ASCII. In ASCII portion unprintable characters are indicated by a period(.).

Secondary memory can be displayed only if it has been frozen by **freeze** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE

```
ds prog, 10, 4        - display 10 long words (4-byte-units)
                        starting from 'prog' memory location

ds 0x040d3448, 10     - display memory starting from the location
                        with the hexadecimal address 0x040d3448. The
                        memory amount and unit size will be taken
                        from the previous ds command.
```

SEE ALSO:

d	- display primary memory.
freeze	- freeze secondary memory.

DISPLAY MFC SIGNAL MONITOR

COMMAND FORMAT:

dsm [*index* | **all**]

OPERANDS:

index - index in the array of the monitored TN(s).

DESCRIPTION:

The **dsm** command allows to display TN being monitored during MFC signalling. If **all** is specified, all monitored TNs will be displayed. If *index* is specified, the TN corresponding to this index in the array of monitored TNs will only be displayed.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
dsm 1
TNM  01  1801
```

```
dsm all
TNM  00  1840
TNM  01  1801
TNM  02  0000
TNM  03  0000
TNM  04  0000
TNM  05  0000
TNM  06  0000
TNM  07  0000
TNM  08  0000
TNM  09  0000
```

SEE ALSO:

csm	- clear MFC signal monitor.
ssm	- set MFC signal monitor.
tsm	- select type of MFC signal monitor message.

ECHO

COMMAND FORMAT:

echo

DESCRIPTION:

The **echo** command prints its arguments onto the standard output. The arguments must be separated by SPACE characters or TAB characters.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXIT

COMMAND FORMAT:

exit

DESCRIPTION:

The **exit** command is used to exit a PDT shell.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

FIND GLOBAL NUMBER

COMMAND FORMAT:

fgn *pc*

OPERANDS:

pc - program counter (hexadecimal).

DESCRIPTION:

The **fgn** command displays the global procedure number and offset within the procedure corresponding to the *pc* value.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

To find the global procedure number for location 1234, the following command should be entered:

```
fgn 4621662
```

As a result, the following information can be displayed:

```
GL= 002 (2) OF= 00000286
```

SEE ALSO:

adr - address.

FORMAT DISK

COMMAND FORMAT:

format /f0 | /f1

OPERANDS:

/f0, /f1 - floppy disk devices.

DESCRIPTION:

The **format** command allows to format a floppy disk and create a file system on it.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

format /f0 - **format** disk on floppy drive 0 and create a file system on it.

FREEZE MEMORY

COMMAND FORMAT:

freeze

DESCRIPTION:

The **freeze** command allows to freeze entire secondary memory (disables write to all secondary memory banks) and thus makes it possible to analyze the memory contents as it was at the moment when some events occurred.

As a result the system switches to non-redundant mode.

This command is available only in debug operation mode.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

```
pdt> freeze
Secondary memory frozen.
System is in non-redundant mode.
pdt>
```

HISTORY

COMMAND FORMAT:

h

DESCRIPTION:

PDT shell has a history mechanism where twenty most recent commands are easily available. This mechanism is similar to the UNIX Korn shell history facility with a built-in line editor that allows previously entered commands to be edited.

The **h** command displays 20 most recent commands entered into the PDT shell; old commands are pushed down as new ones are entered. To edit a command, type ESC to enter the edit mode and use the commands listed below. The ESC key switches the shell to the edit mode. The RETURN key always gives the line to the shell from both editing and input modes.

The following list is a summary of the vi-like commands available in edit mode.

Movement and search commands:

*n***G**	- go to command number *n*.
/s	- search for string *s* backward in history.
?*s*	- search for string *s* forward in history.
n	- repeat last search.
N	- repeat last search in opposite direction.
*n***k**	- get *n*th previous shell command in history.
*n***-**	- same as **k**.
*n***j**	- get *n*th next shell command in history.
n+	- same as **j**.
*n***h**	- move left *n* characters.
^H	- same as **h**.
*n***l**	- move right *n* characters.
SPACE	- same as **l**.
*n***w**	- move *n* words forward.
*n***W**	- move *n* blank-separated words forward.
*n***e**	- move to end of the *n*th next word.
*n***E**	- move to end of the *n*th next blank-separated word.
*n***b**	- move back *n* words.
*n***B**	- move back *n* blank-separated words.
f*c*	- find character *c*, searching forward.
F*c*	- find character *c*, searching backward.
^	- move cursor to first non-blank character in line.
$	- go to end of line.
0	- go to beginning of line.

Insert commands (input is expected until an ESC is typed):

a	- append.
A	- append at end of line.
c SPACE	- change character.
cl	- change character.
cw	- change word.
cc	- change entire line.
c$	- change everything from cursor to end of line.
C	- same as **c$**.
S	- same as **cc**.
i	- insert.
I	- insert at beginning of line.
R	- type over characters.

Editing commands:

*n***r***c*	- replace the following *n* characters with *c*.
*n***x**	- delete *n* characters starting at cursor.
*n***X**	- delete *n* characters to the left of the cursor.
d SPACE	- delete character.
dl	- delete character.
dw	- delete word.
dd	- delete entire line.
d$	- delete all from cursor to end of line.
p	- put last deletion after the cursor.
P	- put last deletion before the cursor.
u	- undo last command.
~	- toggle case, lower to upper or vise versa.

Special commands:

^U	- delete line and leave edit mode.
^L	- redraw line.
^D	- complete symbol name.
RETURN	- give line to shell and leave edit mode.

The default value for *n* is 1.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

BUGS:

The **h** command does not work properly if it is used immediately after shell start (if there is no command in the history stack).

HOSTS

COMMAND FORMAT:

hosts

DESCRIPTION:

The **hosts** command prints a list of all known hosts on the network. For each host a single line should be given with the following information:

```
hostname      Internet address     aliases
```

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

INFORMATION (TASKS SUMMARY)

COMMAND FORMAT:

i

DESCRIPTION:

The **i** command prints a summary of TCB (Task Control Block) for each task in the system. The **ti** command provides more complete information on a specific task.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> i
  NAME        ENTRY        TID     PRI   STAT     PC        SP        ERRN DELAY
--------------- -------------------------------------- ---------- ------------ ------------- --------- ----------
tExcTask    _excTask     429b5a0  0    PEND   40b0a4c  429b504  d0003   0
tLogTask    _logTask     42922f8  0    PEND   40b0a4c  4292258      0   0
tNetTask    _netTask     4252164  50   PEND   40d584e  425210c      0   0
tRstTask    _rstMainTask 4296194  60   PEND   40b0a4c  4296110      0   0
hiserv0     _hiJobServer 4287164  60   PEND   40b0a4c  4287048      0   0
cnipMon     _hiPJobServe 41fcba0  60   DELAY  40d0070  41fcb30      0   2811
ipbMoni     _hiPJobServe 41fa240  60   DELAY  40d0070  41fa1d0      0   466
tPortmapd   _portmapd    424d6c4  100  PEND   40d584e  424d57c     16   0
tSL1        _sl1Main     4710000  240  READY  46218f6  4717f50  3d0002  0
pdtShell01  40899b2      41eb48   240  READY  4046ad6  41ea858  3d0002  0
```

SEE ALSO:

ti	- task information.
tt	- task trace.

INIT INFORMATION

COMMAND FORMAT:

ini

DESCRIPTION:

The **ini** command displays the stored information from the most recent trap.

Note. In earlier versions of release 18, the **ini** command does not work properly. All relevant information can be obtained from the trap data block by means of the **tdb** command, and from report file by means of the **rds** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

tdb - print trap data block.

INDIRECT PRINT

COMMAND FORMAT:

ipr *level offset addr* [*count*]

OPERANDS:

level	- referencing level (hexadecimal).
*offset*s	- two hexadecimal offsets packed into a word by byte(*offset*2 *offset*1).
addr	- memory address to start from (hexadecimal).
count	- number of words to be printed (hexadecimal).

DESCRIPTION:

The **ipr** commands prints virtual memory contents referenced indirectly through a specified pointers chain. The pointers chain starts from the *addr* and its lengths is defined by the *level*. The offset will be added to the effective address to find the next pointer or the data. If the specified *level* is greater than 2, *offset2* will be applied repeatedly. The *count* parameter specifies the amount of data to be printed.

Output will contain pointers chain and the data referenced by the final pointer.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE

```
ipr 3 0204 8013 6

02928E 02921E 0000A6
000000A8 :   00000000 00000000 00000000 00000000 00000000 00000000
```

SEE ALSO:

d, ds	- display real memory.
p, prp, prs	- display virtual memory.

LIST DISASSEMBLED CODE

COMMAND FORMAT:

l [*addr* [, *count*]]

OPERANDS:

addr	- memory address where to start disassembling.
count	- number of instructions to disassemble.

DESCRIPTION:

The **l** command disassembles a specified number of instructions and displays them on standard output device. If the address of an instruction is entered in the system symbol table, the symbol will be displayed as a label for that instruction. Also, addresses in the op-code field of instructions will be displayed symbolically.

Processing will start at the location specified by the *addr* and will continue until the *count* of instructions has been disassembled.

If *addr* is not specified, disassembling will start from the location where the previous **l** command has terminated.

If *count* is not specified, the last specified count is used (initially 10).

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

`l aFunc`	- disassemble a default number of instructions starting from the memory location corresponding to 'aFunc' symbol

SEE ALSO:

d	- display memory contents.
ds	- display secondary memory contents.

LABEL

COMMAND FORMAT:

label *disk-device* [, *label*]

OPERANDS:

disk-device	- floppy disk device (/f0 or /f1).
label	- up to 11 characters, which will be used as a volume label. All characters acceptable in file names are acceptable in the volume label.

DESCRIPTION:

The **label** command allows to create, display and change volume label on a floppy disk.

If *label* is specified, the existing volume label accompanied by a prompt to confirm label change will be displayed. If label change is confirmed, new label will replace the existing one.

If *label* is not specified, the existing volume label will be displayed.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

To change volume label to ‘thor-1’ on a floppy disk on the /f0 device, the following command should be entered:

```
label /f0, thor-1
```

As a result, the following prompts will be displayed:

```
Volume in drive '/f0' has no label
Volume label will be changed to 'thor-1' (y/n)?
```

To confirm change, ‘y’ should be entered.

LIST FILES

COMMAND FORMAT:

ls [*filename1*[, *filename2*...]]

ll [*filename1*[, *filename2*...]]

OPERANDS:

filename	- name of a file or directory. Global characters '?' and '*' can be used to specify f*ilename*.

DESCRIPTION:

ls and **ll** commands allows to print contents of a directory. **ls** provides short format and **ll** provides long format of a file list. In addition to the file name, long format also contains the file size and date and time when file was created.

For each *filename* which is a directory, **ls** lists the contents of the directory.

For each *filename* which is a file, **ls** prints its name. If filename contains global characters, all corresponding files will be listed.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLES:

ll	- list all current directory files in long format.
ls /u/rpt/*.log	- list all files with the extension log in the /u/rpt directory.

SHOW MEMORY ALLOCATION

COMMAND FORMAT:

mem

DESCRIPTION:

The **mem** command shows the system memory partition blocks and statistics. It displays all the blocks in the free list, the total amount of free space, the number of blocks, the average block size, and the maximum block size. It also shows the number of blocks currently allocated, and the average allocated block size.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> mem
FREE LIST:
  num     addr      size
  --- ---------- ----------
    1  0x420ae14         24
    2  0x423596c         28
    3  0x4235dc4         16
    4  0x42743b4         16
    5  0x42732c4         16
    6  0x4273bac         16
    7  0x427501c         16
    8  0x410e1e0     904152
SUMMARY:
 status   bytes    blocks   ave block  max block
 ------ --------- -------- ---------- ----------
current
   free    904316       10      90431    904152
  alloc    741772     1503        493         -
cumulative
  alloc   1309296     1848        708         -
pdt>
```

SEE ALSO:

mstat - memory status.

MAKE DIRECTORY

COMMAND FORMAT:

mkdir *directory* [, *directory*...]

OPERANDS:

directory	- new directory pathname.

DESCRIPTION:

The **mkdir** command allows to create new directories. Standard entries, ‘.’, for the *directory* itself, and ‘..’ for its parent are made automatically.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

`mkdir tmpdir`	- make new directory ‘tmpdir’ as a subdirectory of the current directory.

SEE ALSO:

cd	- change a directory.
rmdir	- remove a directory.

MONITOR

COMMAND FORMAT:

mon [**on**] *addr* [*option*] [*target_value*]

mon off

mon

OPERANDS:

addr	- memory location to be monitored.
option	- action to be taken when the monitored location is changed. The following actions can be specified: 'r' - restore. Restore the monitored location to old contents; 't' - trap. Cause call processing restart (same as **trp** command).
target_value	- value to be compared with the monitored location value.

DESCRIPTION:

The **mon** command allows to monitor a specified location in memory.

When the monitor is active, it looks for a change in *addr* at the end of every timeslice. If a change is detected, the old and new values are displayed.

If *option* is specified, an appropriate action will be taken when the monitored location is changed.

target_value parameter provides a conditional monitoring facility. If *target_value* is specified, the monitor will assume that a change occurs only when the new value in the monitored location matches the *target_value*.

The **mon off** command turns monitoring off. This is the only way the monitor can be turned off.

If no parameters are specified, the current monitor status will be displayed.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

MEMORY STATUS

COMMAND FORMAT:

mstat

DESCRIPTION:

The **mstat** command shows the system memory operation mode (redundant / non-redundant), and status of all memory banks on both sides. This command is especially helpful in conjunction with **freeeze** and **unfreeze** commands.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> mstat
System mode:      Redundant.
   Side    Bank    Address range    Status   Read   Write
--------- ------ ----------------- -------- ------ -------
primary      1   4000000 - 43fffff   enbl    enbl   enbl
primary      2   4400000 - 47fffff   enbl    enbl   enbl
primary      3   4800000 - 4bfffff   enbl    enbl   enbl
primary      4   4c00000 - 4ffffff   enbl    enbl   enbl
primary      5   5000000 - 53fffff   enbl    enbl   enbl
primary      6   5400000 - 57fffff   enbl    enbl   enbl
primary      7   not populated
primary      8   not populated
secondary    1   4000000 - 43fffff   enbl    enbl   enbl
secondary    2   4400000 - 47fffff   enbl    dsbl   dsbl
secondary    3   4800000 - 4bfffff   enbl    enbl   enbl
secondary    4   4c00000 - 4ffffff   enbl    enbl   enbl
secondary    5   5000000 - 53fffff   enbl    enbl   enbl
secondary    6   5400000 - 57fffff   enbl    enbl   enbl
secondary    7   not populated
secondary    8   not populated
```

SEE ALSO:

mem	- show memory allocation.

MOVE FILES

COMMAND FORMAT:

mv *filename1*, *filename2*

mv *filename1*[, *filename2*...] *directory*

OPERANDS:

filename	- name of the file to be copied.
directory	- pathname of the destination directory.

DESCRIPTION:

The first form of **mv** command moves *filename1* to *filename2.* The **mv** command does not rename files when both files (source and destination) are in the same directory. Instead, it copies source file onto destination file and then deletes source file.

The second form moves each *filename* to the indicated *directory* with its original name.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

`mv prog.abc, example`	- move file 'prog.abc' from the current directory onto the file 'example' in the same directory.
`mv /p/prog.abc, /p/example .`	- move files 'prog.abc' and 'example' from the '/p' directory to the current directory.

SEE ALSO:

cp	- copy files.
ren	- rename file.

NETWORK CONTROL MEMORY

COMMAND FORMAT:

net [*loop*] [c]

OPERANDS:

loop	- loop number (hexadecimal).
c	- suppress complementing of data.

DESCRIPTION:

The **net** command reads and displays complement of data for all timeslots for the specified *loop*. The c parameter allows to suppress complementing of data.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

To print the network control memory for loop 0, the following command should be entered:

```
pdt> net 0
```

```
8000: * 4000 B600 * 4000 CC00 * 4000 7F00 * 4000 C300 *
8000: * 4900 B600 * 4000 4A00 * 4000 4400 * 4000 4700 *
8000: * 4000 A300 * 4000 5100 * 4000 AA00 * 4000 5800 *
8000: * 4000 7B00 * 4000 8D00 * 4000 8E00 * 4000 AC00 *
8000: * 4000 1800 * 4000 FF00 * 4000 2300 * 4000 1100 *
8000: * 4000 0A00 * 4000 D800 * 4000 FB00 * 4000 C500 *
8000: * 4000 8100 * 4000 D200 * 4000 F000 * 4000 CB00 *
8000: * 4000 A500 * 4000 9600 * 4000 3400 * 4000 F700 *
```

SEE ALSO:

net	- print network information.

NETWORK INFORMATION

COMMAND FORMAT:

nwk *loop* [*timeslot*]

OPERANDS:

loop	- loop number (hexadecimal).
timeslot	- timeslot number.

DESCRIPTION:

The **nwk** command displays the loop type, status, and timeslot information for the specified *loop* and *timeslot*. The output format depends on the loop type:

Loop Type	Display Format
CONF	
TERM	<timeslot> <NWK extgroup bit> <shelf> <card> <unit> <NWK LINK>
TONE	<timeslot> <tone type> <TONE> <RINGCODE> <TONE DIGIT>
MF	<timeslot> <MF status> <MF state> <MF POINTER> <DIGITS> bits 8-11

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

net - print network control memory.

PRINT VIRTUAL MEMORY

COMMAND FORMAT:

p *addr* [*count*]

OPERANDS:

addr	- memory address to start from (hexadecimal).
count	- number of words to be printed(hexadecimal).

DESCRIPTION:

The **p** command displays the contents of the specified *count* of consecutive locations in virtual memory, starting from the *address*. If *count* is omitted, one location will be displayed.

When the contents of the next line to be printed is identical to the last printed line, the printing of this line is suppressed and a 'S' is placed at the beginning of the next line. (The first line and the last line are always printed.)

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
p 141df 5
000141DF :  0001C62E  0001E8A1 002F4316 0001E8EC 002F42FC
```

SEE ALSO:

d	- display primary memory (real).
ds	- display secondary memory (real).
prp	- print primary memory (virtual).
prs	- print secondary memory (virtual).

PRINT PRIMARY MEMORY

COMMAND FORMAT:

prp *addr* [, *amount*]

OPERANDS:

addr	- memory address to start from.
amount	- number of words to be printed.

DESCRIPTION:

The **prp** command displays a specified *amount* of the consecutive locations in primary virtual memory, starting from the *address*.

If *addr* is not specified, the location where the previous **prp** command has terminated will be used. If *amount* is not specified, its value will be taken from the previous **prp** command.

Memory is displayed in two formats: hexadecimal and ASCII. In ASCII portion, unprintable characters are indicated by a period(.).

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> prp 8 11
00000008:  00000000 00000000  00007ede 00000000  *..........~^....*
0000000c:  00000000 00000000  00000416 002f6688  *............./f.*
00000010:  0001f2f1 002f6788  0001f263           *..rq./g...rc    *
pdt>
```

SEE ALSO:

d	- display primary memory (real).
ds	- display secondary memory (real).
p	- print virtual memory.
prs	- print secondary memory (virtual).

PRINT SECONDARY MEMORY

COMMAND FORMAT:

prs *addr* [, *amount*]

OPERANDS:

addr	- memory address to start from.
amount	- number of words to be printed.

DESCRIPTION:

The **prs** command displays a specified *amount* of the consecutive locations in secondary virtual memory, starting from the *address*.

If *addr* is not specified, the location where the previous **prs** command has terminated will be used. If *amount* is not specified, its value will be taken from the previous **prs** command.

Memory is displayed in two formats: hexadecimal and ASCII. In ASCII portion, unprintable characters are indicated by a period(.).

Secondary memory can be displayed only if it has been frozen.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

```
pdt> prs 100 10
00000100:  00000000 00000000  00007ede 00000000  *..........~^....*
00000104:  00000000 00000000  00000000 00000000  *................*
00000108:  0001f2f1 002f6788                     *..rq./g.        *
pdt>
```

SEE ALSO:

d	- display primary memory (real).
ds	- display secondary memory (real).
p	- print virtual memory.
prp	- print primary memory (virtual).

PRINT WORKING DIRECTORY

COMMAND FORMAT:

pwd

DESCRIPTION:

The **pwd** command prints the pathname of the current (working) directory.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

SEE ALSO:

cd	- change working directory.

REBOOT

COMMAND FORMAT:

reboot [*start_type*]

OPERANDS:

start_type - type of system restart.

DESCRIPTION:

The **reboot** command causes system restart. If *start_type* is -1, COLD system start will be initiated, otherwise WARM restart will be performed.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

RENAME FILE

COMMAND FORMAT:

ren *filename1*, *filename2*

OPERANDS:

filename	- name of a file to be copied.
directory	- pathname of the destination directory.

DESCRIPTION:

The **ren** command changes the name of *filename1* to *filename2*. Changing of the name is allowed only if *filename2* does not already exist.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

ALIAS: **rename**

EXAMPLE:

ren prog.new, prog.old	- rename file 'prog.new' in the current directory to 'prog.old'.

SEE ALSO:

cp	- copy files.
mv	- move files.

REMOVE FILES

COMMAND FORMAT:

rm *filename1*[, *filename2*...]

OPERANDS:

filename	- name of a file to be deleted.

DESCRIPTION:

The **rm** allows to remove disk files.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

ALIAS: **del**

EXAMPLE:

rm prog.abc, /u/example	- remove file prog.abc from current directory and file example from /u directory.

SINGLE-STEP

COMMAND FORMAT:

s [*task*]

OPERANDS:

task	- task to be continued.

DESCRIPTION:

The **s** command allows to single-step a task that is stopped at a breakpoint. The task will execute one instruction, then display the task registers and the next instruction to be executed. If *task* is omitted, the last task referenced is assumed.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

s	- single-step the last referenced task.

SEE ALSO:

c	- continue task execution.
cret	- continue until return.
so	- step over a subroutine.

SL1 INPUT

COMMAND FORMAT:

sl1input

DESCRIPTION:

The **sl1input** command allows to switch the current PDT terminal to SL-1. As a result, input from this terminal will go to SL-1. However, output from both SL-1 and PDT will still be directed to this terminal.

To switch terminal back to PDT, use **^P^D^T**.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1input
OVL111 000 IDLE
TTY 02 SCH MTC TRF BUG    2:05
logi
PASS?
.
>ld 88
AUTH000
MEM AVAIL: (U/P): 2873050    USED: 174373    TOT: 3047423
SCH5066
REQ ^P^D^T
pdt>
```

SL1 BROADCASTING

COMMAND FORMAT:

sl1listen [**on** | **off**]

DESCRIPTION:

The **sl1listen** command enables (**on**) and disables (**off**) SL-1 broadcasting to the current PDT terminal. If no parameter is specified, **on** is assumed.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

SL1 LOAD

COMMAND FORMAT:

sl1load [*sl1res*, *ovlres*]

OPERANDS:

sl1res	- call processing executable module file name.
ovlres	- overlays executable module file name.

DESCRIPTION:

The **sl1load** command loads SL1 executable modules "sl1res" and "ovlres" into the main memory. If no arguments are specified, the modules will be loaded from "/p" device (protected hard disk partition).

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1load
Loading 'res' from "/p/sl1/sl1res" ...
Loading 'ovl' from "/p/sl1/ovlres" ...
```

SEE ALSO:

sl1run - start SL1 task execution.

SL1 RUN

COMMAND FORMAT:

sl1run

DESCRIPTION:

The **sl1run** command initiates Sl1 task execution.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

SEE ALSO:

sl1load - load SL1 code into the memory.

SL1 SPY START

COMMAND FORMAT:

sl1Spy [*period*]

OPERANDS:

period	- sampling frequency, in seconds.

DESCRIPTION:

The **sl1Spy** command starts periodic SL1 work-load data collecting and printing. Data are gathered every *period* of seconds and stored in the SL1 Spy internal queue. To stop data printing (but not collecting), use the **sl1SpyHide** command. To resume data printing, use the **sl1SpyWatch** command. To stop data collecting, use the **sl1SpyStop** command

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1Spy
    Starting sl1Spy. Sampling every 30 secs.
    To stop: use 'sl1SpyStop'
    To hide: use 'sl1SpyHide'
task spawned: id = 0x4186d98, name = t1
value = 0 = 0x0
pdt>
```

SEE ALSO:

sl1SpyHide	- stop SL1 Spy data printing.
sl1SpyStop	- stop SL1 Spy data collecting.
sl1SpyTail	- print SL1 Spy data samples that have already been collected.
sl1SpyWatch	- resume SL1 Spy data printing.

SL1 SPY HIDE

COMMAND FORMAT:

sl1SpyHide

DESCRIPTION:

The **sl1SpyHide** stops printing data started by the **sl1Spy** or **sl1SpyWatch** command. Data will still be collected, but not printed.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1SpyHide
value = 0 = 0x0
pdt>
```

SEE ALSO:

sl1Spy	- start periodic SL1 work-load data collecting.
sl1SpyStop	- stop SL1 Spy data collecting.
sl1SpyTail	- print SL1 Spy data samples that have already been collected.
sl1SpyWatch	- resume SL1 Spy data printing.

SL1 SPY STOP

COMMAND FORMAT:

sl1SpyStop

DESCRIPTION:

The **sl1SpyStop** command stops SL1 Spy data collection.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1SpyStop
sl1Spy task completed.
value = 0 = 0x0
pdt>
```

SEE ALSO:

sl1Spy	- start periodic SL1 work-load data collecting.
sl1SpyHide	- stop SL1 Spy data printing.
sl1SpyTail	- print SL1 Spy data samples that have already been collected.
sl1SpyWatch	- resume SL1 Spy data printing.

l SL1 SPY TAIL

COMMAND FORMAT:

sl1SpyTail *N*

OPERANDS:

N - number of reports (samples) to be printed.

DESCRIPTION:

The **sl1SpyTail** command allows to print last *N* data samples have already been collected by SL1 Spy. Each data sample includes:

- side where data has been collected (i.e. SPY 0), sample number and time stamp;
- current and average number of interrupts for each group;
- current and average value of call processing Work Counts;
- current and average number of entries in call processing queues.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> sl1SpyTail 2
SPY 0: sample 16 (16/3/93 17:13:19)
 Interrupts        READY            IO           LINT
         0:           3(   262)      0(     0)      1(     3)
         4:          29(    21)      0(     0)      0(     4)
 WorkCounts
      Wdog:       72432(      70606)
      Idle:    25869120(   26478720)
 WorkQueues
   Cadence:           2(   queue  2)
      Dial:           1(   queue  6)
      Idle:         194(   queue 12)

SPY 0: sample 17 (16/3/93 17:13:29)
 Interrupts        READY            IO           LINT
         0:           3(   247)      0(     0)      1(     3)
         4:         108(    26)      0(     0)      0(     4)
 WorkCounts
      Wdog:       72150(      70697)
      Idle:    25869120(   26442840)
 WorkQueues
   Cadence:           2(   queue  2)
      Idle:         195(   queue 12)
value = 0 = 0x0
pdt>
```

SEE ALSO:

sl1Spy	- start periodic SL1 work-load data collecting and printing.
sl1SpyHide	- stop SL1 Spy data printing.
sl1SpyStop	- stop SL1 Spy data collecting.
sl1SpyWatch	- resume SL1 Spy data printing.

SL1 SPY WATCH

COMMAND FORMAT:

sl1SpyWatch

DESCRIPTION:

The **sl1SpyWatch** command resumes printing upcoming data samples that will be gathered by SL1 Spy. Each data sample includes:

- side where data has been collected (i.e. SPY 0), sample number and time stamp;
- current and average number of interrupts for each group;
- current and average value of call processing Work Counts;
- current and average number of entries in call processing queues.

Printing can be stopped any time by means of **sl1SpyHide** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **superuser**

EXAMPLE:

```
-> sl1SpyWatch 2, 1
  SL1 SPY Data will be presented 2 times
  To stop Data output use 'sl1SpyWatch 1'.
  The most current SL1 SPY Record:

SPY 0: sample 14 (16/3/93 17:13:00)
 Interrupts        READY            IO          LINT
        0:        1160(   269)      0(     0)      2(     3)
        4:           0(    22)      0(     0)      0(     5)
 WorkCounts
      Wdog:       67252(      70750)
      Idle:    26732280(   26565720)
 WorkQueues
   Cadence:           2(   queue  2)
      Idle:         195(   queue 12)
```

```
SPY 0: sample 15 (16/3/93 17:13:10)
 Interrupts        READY             IO            LINT
         0:          422(   280)      0(     0)      8(     4)
         4:            3(    20)      0(     0)      0(     4)
 WorkCounts
      Wdog:       66767(      70485)
      Idle:    25869120(   26519280)
 WorkQueues
   Cadence:            2(   queue  2)
      Idle:          195(   queue 12)

SPY 0: sample 16 (16/3/93 17:13:19)
 Interrupts        READY             IO            LINT
         0:            3(   262)      0(     0)      1(     3)
         4:           29(    21)      0(     0)      0(     4)
 WorkCounts
      Wdog:       72432(      70606)
      Idle:    25869120(   26478720)
 WorkQueues
   Cadence:            2(   queue  2)
      Dial:            1(   queue  6)
      Idle:          194(   queue 12)

SL1 SPY Data Watch has been completed
  To stop data collection - use 'sl1SpyStop'
  To resume data watch    - use 'sl1SpyWatch N'
->
```

SEE ALSO:

sl1Spy	- start periodic SL1 work-load data collecting.
sl1SpyHide	- stop SL1 Spy data printing.
sl1SpyStop	- stop SL1 Spy data collecting.
sl1SpyTail	- print SL1 Spy data samples have already been collected.

SLIP BEGIN (THOR)

COMMAND FORMAT:

slipBegin

DESCRIPTION:

The **slipBegin** command activates SLIP connection. It puts the current active THOR serial port into SLIP mode.

If **slipBegin** finishes successfully, it prompts a command to be executed from a UNIX terminal to establish SLIP connection to the THOR machine.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> slipBegin
SLIP Gateway (slattach <tty> 100.1.1.1 thor <baudrate>)
```

This **slattach** command is obsolete, please use the **slipBegin** command with the proper tty name and baudrate value to establish SLIP connection to the THOR machine.

SEE ALSO:

slipBegin (THOR)	- start THOR side SLIP mode connection.
slipEnd (THOR/UNIX)	- deactivate appropriate SLIP mode connection.

SLIP BEGIN (UNIX)

COMMAND FORMAT:

slipBegin *<Suntty><baudrate>*

DESCRIPTION:

The **slipBegin** command activates SLIP connection. It puts the selected UNIX serial port into SLIP mode.

If **slipBegin** finishes successfully, it outputs the following:

[*<Shell Job #>*] *<Process ID>*
```
SLIP has been started up.
```

If you do not supply the appropriate parameters, you will get the following:

```
slipBegin <tty> <baud rate>
```

OPERATION MODE: **UNIX Command**

EXAMPLE:

```
pdt> slipBegin
SLIP Gateway (slattach <tty> 100.1.1.1 thor <baudrate>)
```

This **slattach** command is obsolete, please use the **slipBegin** command with the proper tty name and baudrate value to establish SLIP connection to the THOR machine.

SEE ALSO:

slipBegin (UNIX)	- start UNIX side SLIP mode connection.
slipEnd (THOR/UNIX)	- deactivate appropriate SLIP mode connection.

SLIP END (THOR)

COMMAND FORMAT:

slipEnd <cr> <cr>

DESCRIPTION:

The **slipEnd** command deactivates SLIP connection. It is very important to enter two <cr> (carriage return) QUICKLY, after the command to make it work properly.

Try to shut down UNIX side quickly as well, so that the two sides are "in sync" as soon as possible.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

slipBegin (THOR/UNIX)	- activate appropriate SLIP mode connection.
slipEnd (UNIX)	- deactivate UNIX side SLIP mode connection.

SLIP END (UNIX)

COMMAND FORMAT:

slipEnd

DESCRIPTION:

The **slipEnd** command deactivates SLIP connection. It is very important that you shut down the THOR side before shutting down the UNIX side, or you will not be ABLE to shut down the THOR side.

If **slipEnd** finishes successfully, it outputs the following:

```
SLIP (<Process ID>) has been shut down.
```

If **slipEnd** realizes that you were not running SLIP, it outputs the following:

```
SLIP not running at the moment.
```

OPERATION MODE: **UNIX Command**

SEE ALSO:

slipBegin (THOR/UNIX)	- activate appropriate SLIP mode connections.
slipEnd (THOR)	- deactivate UNIX side SLIP mode connection.

STEP OVER A SUBROUTINE

COMMAND FORMAT:

so [*task*]

OPERANDS:

task	- task to be continued.

DESCRIPTION:

The **so** command allows to single-step a task that is stopped at a breakpoint. However, if the next instruction is a JSR or BSR, it breaks at the instruction following the subroutine call instead. If *task* is omitted, the last task referenced is assumed.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

so	- single-step the last referenced task, but step over a subroutine if the next instruction is a subroutine call.

SEE ALSO:

c	- continue task execution.
cret	- continue until return.
s	- single-step a task.

SET MFC SIGNAL MONITOR

COMMAND FORMAT:

ssm [*tn*]

OPERANDS:

tn	- terminal number. Can be specified as a hexadecimal number or in the form of l s c u.

DESCRIPTION:

The **ssm** command stores the *tn* to be monitored during MFC signalling. All terminal numbers to be monitored are stored as elements of the array and the array index assigned to the specified *tn* will be reported as a result of **ssm** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
ssm 1840
TNM  00  1840

ssm 24 0 0 1
TNM  01  1801
```

SEE ALSO:

csm	- clear MFC signal monitor.
dsm	- display TN(s) being monitored during MFC signalling.
tsm	- select type of MFC signal monitor message.

SUPERUSER MODE

COMMAND FORMAT:

su

DESCRIPTION:

The **su** command switches PDT shell operation into a superuser mode. In the superuser mode, PDT invokes VxWorks native shell and operates in its context until an **exit** command is entered. All VxWorks shell functionality is available in this mode.

In the superuser mode, shell prompt is changed. Instead of 'pdt>', prompt '->' is used.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

exit	- exit PDT shell or superuser mode.

SYMBOL TABLE LOAD

COMMAND FORMAT:

symload [*symfile*]

OPERANDS:

symfile - name of the file with symbol definitions.

DESCRIPTION:

The **symload** command adds symbol definitions from the *symfile* to the system symbol table. If file name is omitted, symbols will be loaded from "/p/sl1/res.sym" file.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

SEE ALSO:

slipBegin- activate SLIP mode connection.

TASK DELETE

COMMAND FORMAT:

td *task*

OPERANDS:

task	- task name or task ID.

DESCRIPTION:

The **td** command allows to delete a task. It makes the specified task exit, and deallocates the stack and memory resources.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLES:

```
td tTask1            - task to delete is specified by name.
td 0x41daed4         - task to delete is specified by task ID.
```

SEE ALSO:

i	- tasks summary.
ti	- task information.

TRAP DATA BLOCK

COMMAND FORMAT:

tdb

DESCRIPTION:

The **tdb** command prints out Trap Data Block (TDB) data. The printout contains the following data:

- data time and context (task or interrupt) when TDB was built;
- task description block;
- task control block (TCB);
- stack contents;
- SL1 unprotected global variables.

Note. In earlier versions of release 18, the **tdb** command does not print formatted return address stack. This information can be obtained from the report file by means of the **rds** command.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

trp - build trap data block and restart call processing task.

TASK INFORMATION

COMMAND FORMAT:

ti [*task*]

OPERANDS:

task - task name or task ID.

DESCRIPTION:

The **ti** command prints the control block (TCB) information for the specified *task*, including registers. If *task* is not specified, the default task is used. The default task is the last task that hits breakpoint or cause an exception, etc.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
ti tSL1
  NAME        ENTRY       TID     PRI   STAT     PC        SP     ERRN  DELAY
--------------- ------------------ ------------- ------- ----------- ------------ ------------- --------- --------
tSL1          _sl1Main       4710000   240    READY  4621662   4717f78   3d0002    0
stack: base 0x4718000  end 0x4710158  size 31956  high 5764   margin 26192
options: 0x10
VX_STDIO
D0 =       0    D4 =       0    A0 =          80      A4 =           0
D1 =       1    D5 =       0    A1 =  569eff0      A5 =  4ab0000     SR =        3019
D2 =       0    D6 =       0    A2 =  4717f98      A6 =  4717f80     PC =  4621662
D3 =       0    D7 =       0    A3 =  4717f80      A7 =  4717f78
```

SEE ALSO:

i - tasks summary.

tt - task trace.

TN TRANSLATION

COMMAND FORMAT:

tnt *tn*

OPERANDS:

tn - terminal number. Can be specified as a hexadecimal number or in the form of l s c u.

DESCRIPTION:

The **tnt** command displays the following pointers associated the specified *tn*:

PGRPPTR, PLPPTR, ULPPTR, PCPTR, UCPTR, PLPTR, ULPTR.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLES:

In the following examples, we use the same terminal number, but it is specified in two different forms:

```
tnt 24 0 0 1

EQPD SLOOP TN 001801
GP 0001C62E SLP 0001E8A1 002F4316  CD 0001E8EC 002F42FC  LN 0001E99B
002F42C4

tnt 1801

EQPD SLOOP TN  24  0  0  1
GP 0001C62E SLP 0001E8A1 002F4316  CD 0001E8EC 002F42FC  LN 0001E99B
002F42C4
```

SEE ALSO:

crg - call register.

TREE

COMMAND FORMAT:

tree *directory*

OPERANDS:

directory	- directory pathname.

DESCRIPTION:

The **tree** command allows to display all of the directory paths found on the specified *directory*.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

tree /u	- list all directory paths in the '/u' directory.

SEE ALSO:

ls, **ll**	- list files.

TRAP

COMMAND FORMAT:

trp

DESCRIPTION:

The **trp** command causes call processing restart. As a result of the execution of the **trp** command, a new trap data block will be build.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

tdb	- print trap data block.
reboot	- restart the system.

TYPE OF MFC MESSAGE

COMMAND FORMAT:

tsm [*type*]

OPERANDS:

type - message type.

DESCRIPTION:

Th**e tsm** command allows to select type of MFC messages to be printed. It is also used to set DTI2 / PRI2 signal monitor. *type* can have one of the following values:

0 - no message printed;
1 - MFC / MFE messages (message words);
2 - MFC / MFE messages (signal levels);
3 - incoming DT2 / PRI2 messages;
4 - outgoing DT2 / PRI2 messages;
5 - received messages causing state change;
6 - sent messages causing state change;
7 - DTI2 / PRI2 audit monitor;
8 - monitor all DTI2 / PRI2 TNs;

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLE:

```
ssm 1840
TNM  00  1840

ssm 24 0 0 1
TNM  01  1801
```

SEE ALSO:

csm - clear MFC signal monitor.

dsm - display TN(s) being monitored during MFC signalling.

ssm - set MFC signal monitor.

TASK TRACE

COMMAND FORMAT:

tt [*task*]

OPERANDS:

task	- task name or task ID.

DESCRIPTION:

The **tt** command allows to print a return address stack trace of the specified *task*. It prints a list of the nested routine calls that the task is in. Each routine call and its parameters are shown.

tt can only trace the stack of a task other than itself. For example, it cannot be used to trace the stack of the PDT shell it was called from.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

```
pdt> tt tSL1

40d62d6 _vxTaskEntry+10: _sl1Main  (0,  0,  &_sl1StackPtr,  4710158,
               4288970, 428896c, &_sl1TCBPtr, 4710000, 4288970, 101)
46d3c50 _sl1Main    +1ba:__WORKSHED   ([0, 0, 0, 40d62d8, 0])
462142a __WORKSHED  +48: _NEXT_TASK   ([0, 4717fd0, 46d3c56, 0, 0])
46d4ce4 _NEXT_TASK  +8e: _get_ssd_msg_addr([0,0,4717fc0,4621432,0])
```

SEE ALSO:

i	- tasks summary.
ti	- task information.

TYPE

COMMAND FORMAT:

type [*filename* [, *filename*]...]

OPERANDS:

filename	- name of a file to be displayed.

DESCRIPTION:

The **type** command allows to display contents of the specified files on the standard output device.

SECURITY LEVEL: **technical assistance, system support**

OPERATION MODE: **regular**

EXAMPLE:

`type mac.x`	- display contents of ‘mac.x’ file on the standard output device

SEE ALSO:

cat	- concatenate and display.

UNFREEZE MEMORY

COMMAND FORMAT:

unfreeze

DESCRIPTION:

The **unfreeze** command allows to unfreeze secondary memory that had been frozen before. It enables write to all secondary memory banks, synchronizes both primary and secondary memory, and switches the system to redundant mode.

The memory synchronization requires some time. Please, BE PATIENT.

This command is available only in debug operation mode.

SECURITY LEVEL: **system support**

OPERATION MODE: **debug**

EXAMPLE:

```
pdt> unfreeze
Secondary memory unfrozen.
System is in redundant mode.
pdt>
```

WRITE TO MEMORY

COMMAND FORMAT:

w *addr* [*data*]

OPERANDS:

addr	- memory address, from where to start writing (hexadecimal).
data	- data to put into the memory.

DESCRIPTION:

The **w** command allows to modify the memory contents. It displays contents of successive memory locations starting with the *address* and provides the following choices after displaying each of them:

- type a new value followed by <sp> to replace the old value and to continue with the successive location;
- type a new value followed by <cr>, to replace the old value and to stop the command;
- type the <cr> only to leave this value unchanged and to stop the command.

If *data* is specified, there will be no interaction, and the *data* will replace the contents of the specified location.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

EXAMPLES:

```
pdt> p 8013 6
00008013: 00014263 001E6A78 001EB2C3 00000000 00000001 00000000
pdt> w 8016
00008016: 00000003 /00000004<sp>
00008017: 00000000 /00000005<sp>
00008018: 00000000 /<cr>
pdt> w 8018 00000006
pdt> p 8013 6
00008013: 00014263 001E6A78 001EB2C3 00000004 00000005 00000006
```

WRITE XNET MEMORY

COMMAND FORMAT:

wxn *loop timeslot word*

OPERANDS:

loop	- superloop number (should be a multiple of 4).
timeslot	- timeslot number.
word	- word number (0 - 3).

DESCRIPTION:

The **wxn** command allows to modify the network control memory. It displays contents of the specified *word* for the specified *loop* and *timeslot* and provides the following choices:

- type a new value followed by <cr>, to replace the old value;
- type <cr> only to leave this value unchanged.

SECURITY LEVEL: **system support**

OPERATION MODE: **regular**

SEE ALSO:

net - print network control memory.

Appendix B SL1 Fast Globals

This appendix includes a list of so-called SL-1 fast global variables.

0008:	CRPTR	Uptr		
0009:	IPPTR	Uptr		
000a:	OUTPTR	Uptr		
000b:	ABS_CARD	int	bo=2,	bw=4
000b:	ABS_GROUP	int	bo=13,	bw=3
000b:	ABS_G_LOOP	int	bo=8,	bw=8
000b:	ABS_LOOP	int	bo=8,	bw=5
000b:	ABS_SC_UNIT	int	bo=0,	bw=8
000b:	ABS_SHELF	int	bo=6,	bw=2
000b:	ABS_S_CARD	int	bo=2,	bw=6
000b:	ABS_UNIT	int	bo=0,	bw=2
000b:	TERMINAL	int		
000c:	SRCPTR	Uptr		
000d:	DESTPTR	Uptr		
000e:	ATTN_ITEM	int		
000f:	ULATTNPTR	Uptr{6}		
0010:	PLATTNPTR	Pptr{e0}		
0011:	UCATTNPTR	Uptr{5}		
0012:	PCATTNPTR	Pptr{de}		
0013:	ULPATTNPTR	Uptr{4}		
0014:	PLPATTNPTR	Pptr{dd}		
0015:	PGRPATTNPTR	Pptr{dc}		
0016:	ORIG_ITEM	int		
0017:	ULORIGPTR	Uptr{6}		
0018:	PLORIGPTR	Pptr{e0}		
0019:	UCORIGPTR	Uptr{5}		
001a:	PCORIGPTR	Pptr{de}		
001b:	ULPORIGPTR	Uptr{4}		
001c:	PLPORIGPTR	Pptr{dd}		
001d:	PGRPORIGPTR	Pptr{dc}		
001e:	ORIGSSD	Uptr{6}		
001f:	TERSSD	Uptr{6}		
0020:	CFPTR	Uptr{4}		
0021:	PCFPTR	Pptr{dd}		
0022:	CRPTR2	Uptr		
0023:	SCPTR1	Uptr		
0024:	SCPTR2	Uptr		
0025:	ATTNDNPTR	Pptr{e1}		
0026:	DNPTR	Pptr{e1}		
0027:	TER_ITEM	int		
0028:	ULTERPTR	Uptr{6}		
0029:	PLTERPTR	Pptr{e0}		
002a:	UCTERPTR	Uptr{5}		
002b:	PCTERPTR	Pptr{de}		
002c:	ULPTERPTR	Uptr{4}		
002d:	PLPTERPTR	Pptr{dd}		
002e:	PGRPTERPTR	Pptr{dc}		
002f:	SRC_ITEM	int		
0030:	ULSRCPTR	Uptr{6}		
0031:	PLSRCPTR	Pptr{e0}		
0032:	UCSRCPTR	Uptr{5}		
0033:	PCSRCPTR	Pptr{de}		
0034:	ULPSRCPTR	Uptr{4}		
0035:	PLPSRCPTR	Pptr{dd}		
0036:	PGRPSRCPTR	Pptr{dc}		
0037:	DEST_ITEM	int		
0038:	ULDESTPTR	Uptr{6}		
0039:	PLDESTPTR	Pptr{e0}		
003a:	UCDESTPTR	Uptr{5}		
003b:	PCDESTPTR	Pptr{de}		
003c:	ULPDESTPTR	Uptr{4}		
003d:	PLPDESTPTR	Pptr{dd}		
003e:	PGRPDESTPTR	Pptr{dc}		
003f:	TTR_ITEM	int		
0040:	ULTTRPTR	Uptr{6}		
0041:	PLTTRPTR	Pptr{e0}		
0042:	UCTTRPTR	Uptr{5}		
0043:	PCTTRPTR	Pptr{de}		
0044:	ULPTTRPTR	Uptr{4}		
0045:	PLPTTRPTR	Pptr{dd}		
0046:	PGRPTTRPTR	Pptr{dc}		
0047:	DIALED_ITEM	int		
0048:	ULDIALEDPTR	Uptr{6}		
0049:	PLDIALEDPTR	Pptr{e0}		
004a:	UCDIALEDPTR	Uptr{5}		
004b:	PCDIALEDPTR	Pptr{de}		
004c:	ULPDIALEDPTR	Uptr{4}		
004d:	PLPDIALEDPTR	Pptr{dd}		
004e:	PGRPDIALEDPTR	Pptr{dc}		
004f:	DIALED_CRPTR	Uptr		
0050:	DIALED_DNPTR	Pptr{e1}		
0051:	PCDATAPTR	Pptr{e4}		
0052:	AUX_PCDATAPTR	Pptr{e4}		
0053:	UCDATAPTR	Uptr{a}		
0054:	HUNT_DNPTR	Pptr{e1}		
0055:	X_ITEM	int		
0056:	ULXPTR	Uptr{6}		
0057:	PLXPTR	Pptr{e0}		
0058:	UCXPTR	Uptr{5}		
0059:	PCXPTR	Pptr{de}		
005a:	ULPXPTR	Uptr{4}		
005b:	PLPXPTR	Pptr{dd}		
005c:	PGRPXPTR	Pptr{dc}		
005d:	Y_ITEM	int		
005e:	ULYPTR	Uptr{6}		
005f:	PLYPTR	Pptr{e0}		
0060:	UCYPTR	Uptr{5}		
0061:	PCYPTR	Pptr{de}		
0062:	ULPYPTR	Uptr{4}		
0063:	PLPYPTR	Pptr{dd}		
0064:	PGRPYPTR	Pptr{dc}		
0065:	DATAPTR	Pptr{e0}		
0066:	URB_PTR	Uptr{8}		
0067:	PRB_PTR	Pptr{e1}		
0068:	RRB_PTR	Pptr{e7}		
0069:	TLIST_PTR	Pptr{e6}		
006a:	TRK_ITEM	int		
006b:	ULTRKPTR	Uptr{6}		

006c:	PLTRKPTR	Pptr{e0}	
006d:	UCTRKPTR	Uptr{5}	
006e:	PCTRKPTR	Pptr{de}	
006f:	ULPTRKPTR	Uptr{4}	
0070:	PLPTRKPTR	Pptr{dd}	
0071:	PGRPTRKPTR	Pptr{dc}	
0072:	DLAMP	int	
0073:	SLAMP	int	
0074:	WORKLOOP	int	
0075:	WORKPTR	Uptr	
0076:	DNXLSTART	Uptr	
0077:	LINK_QUEUE_PTR	Uptr	
0078:	LINK_F_QUEUE_PTR	Uptr	
0079:	REMOVE_QUEUE_PTR	Uptr	
007a:	LINK_BLOCK_PTR	Uptr	
007b:	LINK_F_BLOCK_PTR	Uptr	
007c:	REMOVE_BLOCK_PTR	Uptr	
007d:	UNLINK_BLOCK_PTR	Uptr	
007e:	FIRSTCRBLOCK_PTR	Uptr	
007f:	BACK_LINK	Uptr	
0080:	RLA_ITEM	int	
0081:	ULRLAPTR	Uptr{6}	
0082:	PLRLAPTR	Pptr{e0}	
0083:	UCRLAPTR	Uptr{5}	
0084:	PCRLAPTR	Pptr{de}	
0085:	ULPRLAPTR	Uptr{4}	
0086:	PLPRLAPTR	Pptr{dd}	
0087:	PGRPRLAPTR	Pptr{dc}	
0088:	TDET_ITEM	int	
0089:	ULTDETPTR	Uptr{6}	
008a:	PLTDETPTR	Pptr{e0}	
008b:	UCTDETPTR	Uptr{5}	
008c:	PCTDETPTR	Pptr{de}	
008d:	ULPTDETPTR	Uptr{4}	
008e:	PLPTDETPTR	Pptr{4}	
008f:	PGRPTDETPTR	Pptr{dc}	
0090:	DLI_ITEM	int	
0091:	ULDLIPTR	Uptr{6}	
0092:	PLDLIPTR	Pptr{e0}	
0093:	UCDLIPTR	Uptr{5}	
0094:	PCDLIPTR	Pptr{de}	
0095:	ULPDLIPTR	Uptr{4}	
0096:	PLPDLIPTR	Pptr{dd}	
0097:	PGRPDLIPTR	Pptr{dc}	
0098:	AWU_ITEM	int	
0099:	ULAWUPTR	Uptr{6}	
009a:	PLAWUPTR	Pptr{e0}	
009b:	UCAWUPTR	Uptr{5}	
009c:	PCAWUPTR	Pptr{de}	
009d:	ULPAWUPTR	Uptr{4}	
009e:	PLPAWUPTR	Pptr{dd}	
009f:	PGRPAWUPTR	Pptr{dc}	
00a0:	WRITE_ITEM	int	
00a1:	ULWRTPTR	Uptr{6}	
00a2:	PLWRTPTR	Pptr{e0}	
00a3:	UCWRTPTR	Uptr{5}	
00a4:	PCWRTPTR	Pptr{de}	
00a5:	ULPWRTPTR	Uptr{4}	
00a6:	PLPWRTPTR	Pptr{dd}	
00a7:	PGRPWRTPTR	Pptr{dc}	
00a8:	MR_ITEM	int	
00a9:	ULMRPTR	Uptr{6}	
00aa:	PLMRPTR	Pptr{e0}	
00ab:	UCMRPTR	Uptr{5}	
00ac:	PCMRPTR	Pptr{de}	
00ad:	ULPMRPTR	Uptr{4}	
00ae:	PLPMRPTR	Pptr{dd}	
00af:	PGRPMRPTR	Pptr{dc}	
00b0:	PRA_ITEM	int	
00b1:	ULPRAPTR	Uptr{6}	
00b2:	PLPRAPTR	Pptr{e0}	
00b3:	UCPRAPTR	Uptr{5}	
00b4:	PCPRAPTR	Pptr{de}	
00b5:	ULPPRAPTR	Uptr{4}	
00b6:	PLPPRAPTR	Pptr{dd}	
00b7:	PGRPPRAPTR	Pptr{dc}	
00b8:	TMP_ITEM	int	
00b9:	ULTMPPTR	Uptr{6}	
00ba:	PLTMPPTR	Pptr{e0}	
00bb:	UCTMPPTR	Uptr{5}	
00bc:	PCTMPPTR	Pptr{de}	
00bd:	ULPTMPPTR	Uptr{4}	
00be:	PLPTMPPTR	Pptr{dd}	
00bf:	PGRPTMPPTR	Pptr{dc}	
00c0:	CDRQ_ITEM	int	
00c1:	CDRQ_ULPTR	Uptr{6}	
00c2:	CDRQ_PLPTR	Pptr{e0}	
00c3:	CDRQ_UCPTR	Uptr{5}	
00c4:	CDRQ_PCPTR	Pptr{de}	
00c5:	CDRQ_ULPPTR	Uptr{4}	
00c6:	CDRQ_PLPPTR	Pptr{dd}	
00c7:	CDRQ_PGRPPTR	Pptr{dc}	
00c8:	KEYNUM	int	
00c9:	KEY0_7	int	
00ca:	SSD_KEY	int	
00cb:	SSDNUM	int	
00cc:	BCS_FUNCTION	int	
00cd:	DIALED_DNDBIT	int	
00ce:	DCH_INTERRUPT	int	bo=15, bw=1
00ce:	DCH_LSTATE	int	bo=10, bw=3
00ce:	DCH_MSTATE	int	bo=8, bw=2
00ce:	DCH_RX_READY	int	bo=13, bw=1
00ce:	DCH_SRD	int	
00ce:	DCH_TX_EMPTY	int	bo=14,bw=1
00cf:	DCHOP	int	

```
00d0:  DCH_OMSG_VARS       int[6]
00d0:  DCH_OMSG_W0         int
00d0:  DCH_W0_PRIMITIVE    int
00d0:  DCH_W0_PRIM_ID      int   bo=1,    bw=7
00d0:  G_MSG_TYPE          int   bo=8,    bw=8
00d0:  G_PRIMIT_ID         int   bo=1,    bw=7
00d1:  DCH_OMSG_W1         int
00d1:  DCH_W1_MAINT_TSK    int
00d1:  DCH_W1_PARM_ID      int
00d1:  G_CREF_FLAG         int   bo=15,  bw=1
00d1:  G_CREF_NUM          int
00d2:  DCH_OMSG_W2         int
00d2:  DCH_W2_PARM_DBUG int   bo=1,    bw=1
00d2:  DCH_W2_PARM_LPBK    int   bo=0,    bw=1
00d2:  G_CH_TN             int
00d2:  G_MSGCRPTR          Uptr
00d2:  G_REF_TAB_INDEX     int
00d3:  DCH_OMSG_W3         int
00d3:  G_CAUSE_OUTP        int   bo=5,    bw=7
00d3:  G_FAC_NO_TN         int   bo=15,  bw=1
00d3:  G_MISC_INF1         int
00d3:  G_RESTART_OUTP      int   bo=12,  bw=3
00d3:  G_STATE_OUTP        int   bo=0,    bw=5
00d4:  DCH_OMSG_W4         int
00d4:  G_IT_ID_OUTP        int   bo=4,    bw=7
00d4:  G_MISC_INF2         int
00d4:  G_NCT_LOCATOR       int
00d4:  G_NSF_NEEDED        int   bo=12,  bw=1
00d4:  G_PROGRESS_OUTP     int   bo=0,    bw=4
00d4:  G_REF_FLAG          int   bo=15,  bw=1
00d4:  G_TNS_NEEDED        int   bo=13,  bw=1
00d4:  G_UUI_NEEDED        int   bo=14,  bw=1
00d5:  DCH_OMSG_W5         int
00d5:  G_CH_NEEDED         int   bo=5,    bw=2
00d5:  G_FAC_NEEDED        int   bo=3,    bw=1
00d5:  G_MAINT_REQ         int   bo=0,    bw=1
00d5:  G_MAINT_STAT        int   bo=1,    bw=2
00d5:  G_NONCALL_REQ       int   bo=4,    bw=1
00d6:  DCH_LOG_NUM         int
00d7:  P_DCH_PTR           Pptr{b8}
00d8:  U_DCH_PTR           Uptr
00d9:  U_BDCH_PTR          Uptr
00da:  P_REF_TAB_PTR       Pptr{b8}
00db:  U_REF_TAB_PTR       Uptr{19}
00dc:  U_MSG_TAB_PTR       Uptr{19}
00dd:  I_BUF_PTR           Uptr
00de:  O_BFR_PTR           Uptr
00df:  P_ESLTN_TABPTR      Pptr{b8}
00e0:  PHY_DCH_PTR         Pptr
00e1:  TIER1_SCHEDULER     int
00e2:  TIER2_SCHEDULER     int
00e3:  THIRTYSECS          int
00e4:  TWOSEC_NUM_MS       int
00e5:  TWOSECONDS          int
00e6:  OVL_TIMER           int
00e7:  WORKCOUNT           int[2]
00e9:  WORKCOUNTCTRL       int
00ea:  HOURCOUNT           int[2]
00ec:  HALFHRCOUNT         int[2]
00ee:  CDR_INHIBIT         int
00ef:  LAMPAUDITPM         int
00f0:  LAMPAUD_INHIBIT     int
```