DM3 Diagnostic Utilities

Reference Guide

05-1484-004

COPYRIGHT NOTICE

All contents of this document are subject to change without notice and do not represent a commitment on the part of Intel Corporation. Every effort is made to ensure the accuracy of this information. However, due to ongoing product improvements and revisions, Intel Corporation cannot guarantee the accuracy of this material, nor can it accept responsibility for errors or omissions. No warranties of any nature are extended by the information contained in these copyrighted materials. Use or implementation of any one of the concepts, applications, or ideas described in this document or on web pages maintained by Intel Corporation may infringe one or more patents or other intellectual property rights owned by third parties. Intel Corporation does not condone or encourage such infringement. Intel Corporation makes no warranty with respect to such infringement, nor does Intel Corporation waive any of its own intellectual property rights which may cover systems implementing one or more of the ideas contained herein. Procurement of appropriate intellectual property rights and licenses is solely the responsibility of the system implementer. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

Publication Date: September 2001

Part Number: 05-1484-004

Intel Corporation
Telecommunications and Embedded Group
1515 Route 10
Parsippany NJ 07054

For **Technical Support**, visit the Dialogic support website at:
http://support.dialogic.com

For **Sales Offices** and other contact information, visit the main Dialogic website at:
http://www.dialogic.com

Table of Contents

List of Figures

List of Tables

About This Information

The following topics provide information about this guide:

- Purpose
- Intended Audience
- How to Use This Information
- Related Information

Purpose

This guide describes the DM3 diagnostic tools and tells you how to use them.

NOTE: The terms “utilities” and “tools” are used interchangeably in this document.

Intended Audience

This information is intended for:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

How to Use This Information

This information is organized as follows:

- Chapter 1, “Overview” provides an overview of the DM3 diagnostic tools and states the purpose of running each one.
- Chapter 2, “System Requirements” provides the hardware and software requirements for using the test utilities.

- Chapter 3, “DM3 Diagnostic Utility Descriptions” gives a description and parameters for each utility. The utilities are listed alphabetically.
- Chapter 4, “Troubleshooting” describes the information to collect if you need to contact Dialogic Technical Support.

Related Information

See the following for more information:

- *System Release Guide* - lists the products and features supported by the system release.
- *System Release Update* - provides last-minute updates not documented in the published documentation, such as known problems and workarounds, and compatibility issues.
- Quick Install Cards - these are shipped with each Dialogic board and they provide installation procedures.
- Software Installation and Configuration Guides - provide information on installing and configuring the System Release software:
  - *System Release Installation and Configuration Guide for Windows*
  - *System Release for Linux Installation Guide*
- FirstCall InfoServer support web site: *http://support.dialogic.com/*

For additional DM3 information, see the following documentation:

- *DM3 Configuration File Reference Guide*
- *Using the DM3 Direct Interface for Windows*
- *DM3 Direct Interface Function Reference for Windows*
- *DM3 Host Library Software Reference for Linux*
- *DM3 Mediastream Architecture Overview*

1. Overview

This guide presents diagnostic utilities for DM3 boards. These utilities can help developers test and debug applications and troubleshoot the system's operation. This overview lists all the utilities and gives general information about the Qscript subset of utilities.

CAUTION

Using some of these utilities involves stopping and resetting boards or changing board parameters (refer to Chapter 3, "DM3 Diagnostic Utility Descriptions" for details). Because using these utilities can disrupt a board's normal operation, they should be used cautiously by experienced personnel only.

1.1. Summary of Test Utilities

Table 1 lists all the DM3 diagnostic utilities, briefly describes each one, and specifies the operating system(s) on which each can be used. Detailed information on each utility is given in Chapter 3, "DM3 Diagnostic Utility Descriptions".

Table 1. List of DM3 Utilities

Utility	Description	Operating System
Alarms	Monitors the alarms on a T-1 or E-1 line.	Windows, Linux
Audio Control	Controls the Player resource and Recorder resource.	Windows, Linux
CallInfo	Detects call information via the TSP resource and lets you specify the call-related events you want to trace.	Windows, Linux

Table 1. List of DM3 Utilities (Continued)

Utility	Description	Operating System
CAS Signal Editor	Allows you to dynamically view and modify CAS signal identification parameters (transitions, pulses, trains, or sequences) so you can test them before changing the *.config* file.	Windows, Linux
DebugView	Lets you monitor debug output on your local system, or any computer on the network that you can reach via TCP/IP.	Windows
Digit Detector	Demonstrates the use of the Tone Generator and Signal Detector components.	Windows, Linux
DM3Insight	Traces and decodes Message and Stream traffic between host and DM3 boards.	Windows
Dm3KDebug	Prints out useful DM3 kernel information.	Windows, Linux
Dm3post	Performs POST diagnostics on demand.	Windows, Linux
Dm3StdErr	Polls the board and posts the qPrintf( ) statements from the resources and DM3 Kernel.	Windows, Linux
Dm3Trace	Prints out qTrace statements based on the qTraceLevel( ) DM3 Kernel function.	Windows, Linux
Getver	Retrieves the version string of a DM3 binary file.	Windows, Linux

Table 1. List of DM3 Utilities (Continued)

Utility	Description	Operating System
ISDN Trace	Provides the ability to track Layer 3 (Q.931) messages on the ISDN D-channel.	Windows, Linux
KernelVersion	Queries the DM3 Kernel for its version number.	Windows, Linux
LineAdmin	Puts lines into service so you can run many of the other diagnostic utilities. Also monitors the alarms on a T-1 or E-1 line.	Windows, Linux
Listboards	Displays status information for baseboard and daughterboards (if any).	Windows, Linux
MercMon	Displays Class Driver and Protocol Driver counter information.	Windows
PDK Configuration Utility	Used when integrating E-1 R2MF protocol information (CDP files) into IPLink, Fax or HDSI FCD files.	Windows, Linux
PDKManager Utility	Used to download and configure E-1 R2MF protocol information on QuadSpan or DualSpan boards.	Windows, Linux
Phone	Provides DM3 GlobalCall Resource call control.	Windows, Linux
QError	Returns the string associated with kernel error codes.	Windows, Linux
STD Config	Configures DM3 component parameters.	Windows, Linux

Table 1. List of DM3 Utilities (Continued)

Utility	Description	Operating System
StrmStat	Displays the status of specified stream(s).	Windows
TSP Config	Changes protocol variant parameters (ISDN and T1 CAS) dynamically.	Windows, Linux
TSP Monitor	Monitors the DM3 GlobalCall Resource protocol.	Windows, Linux
TSP Tracer	Traces CAS protocol operations and includes timing information.	Windows, Linux

1.2. QScript Utilities

The QScript utilities are a subset of the DM3 diagnostic utilities. QScript is an object-oriented scripting tool developed by Dialogic for the DM3 product family. QScript is intended for use while developing demonstration or test programs and is implemented using the Tcl/Tk generic scripting language. All QScript utilities can be run from Linux and Windows operating systems.

QScript utilities use board and line numbers as follows: board numbers are 0-based and line numbers are 1-based. That is, the first board is typically board 0 and the first line is line 1.

The following are QScript utilities:

- Alarms
- Audio Control
- CallInfo
- CAS Signal Editor
- Digit Detector
- LineAdmin
- PDK Configuration Utility

- PDKManager Utility
- Phone
- STD Config
- TSP Config
- TSP Monitor
- TSP Tracer

These utilities are fully described in Chapter 3, “DM3 Diagnostic Utility Descriptions”, where they are listed in alphabetical order among all the other DM3 diagnostic utilities. For information on environment variables and the location of batch or script files, see Section 2.2.3, “QScript Utilities Requirements”, on page 8.

2. System Requirements

To run DM3 diagnostic utilities, you need to address:

- Hardware Requirements
- Software Requirements

2.1. Hardware Requirements

The following hardware is required:

- A Dialogic DM3 board
- A 200 or 233 MHz Pentium PC or better
- At least 64 Mb RAM for a system that contains up to two DM3 boards and 128 Mb RAM for a system that contains more than two DM3 boards

Refer to the *Dialogic System Release Guide* and *Dialogic System Release Update* for the most up-to-date information on board support for these utilities.

2.2. Software Requirements

Software requirements for the DM3 diagnostic utilities include:

- Operating System Requirements
- Dialogic System Release Requirements
- QScript Utilities Requirements

2.2.1. Operating System Requirements

DM3 diagnostic utilities are available for the following operating systems:

- Windows NT 4.0
- Windows 2000
- Linux

NOTE: The term Windows includes both Windows NT and Windows 2000 operating systems.

2.2.2. Dialogic System Release Requirements

A Dialogic System Release must be installed in order to access the DM3 diagnostic tools. See the *Dialogic System Release Guide* to verify that the tools are supported for a particular release.

It is recommended that Dialogic software be installed in the default installation directory under all operating system environments. In Windows, the default install location is:

```
%systemroot%\program files
```

In Linux, the default install location is:

```
/usr/dialogic
```

2.2.3. QScript Utilities Requirements

The following information gives details specific to QScript utilities:

- File Directories
- QScript Environment Variables

File Directories

The directory for the batch file (Windows) or script file (Linux) used to invoke the QScript tools is:

- Windows: %systemroot%\program files\dialogic\bin
- Linux: /usr/dialogic/bin

The QScript tools developed by Dialogic are located in:

- Windows: %systemroot%\program files\dialogic\qscript
- Linux: /usr/dialogic/qscript

NOTE: Do not run a <*toolname*>*.qs* file directly. Batch or script files have been created which call the QScript interpreter to run the <*toolname*>*.qs* file. To use a QScript utility, specify the utility name and parameters on the command line.

QScript Environment Variables

QSCRIPT_DIR Environment Variable:

To run the QScript tools in a Linux environment, the QSCRIPT_DIR environment variable needs to be set. In a Windows environment, this variable is set during the Dialogic System Release install. In Linux environments, set QSCRIPT_DIR to:

```
$DLGCROOT/qscript
```

Single Session Variable:

In a Windows environment, set the variable for a single session using the `set` command. To permanently set the environment variable for all login sessions, update the variable in the **System Properties Environment** tab.

In a Linux environment, set the variable for a single session using the `setenv` command (C Shell) or `set and export` command (K and Bourne Shell). To permanently set the environment variable for all login sessions, update the user's *.profile* file to include the variable and associated values.

Remote Systems:

The remote system containing the board does not need to have QScript installed, but must be running the RemoteQHostServer application included with QScript and installed in the bin directory.

To run QScript tools against a board in a remote system, set the REMOTE_QHOST environment variable to the name of the machine that contains the board you want to access. Set REMOTE_QHOST to:

```
hostname:port
```

where hostname is the machine name or TCP/IP address, and port is optional and specified only if RemoteQHostServer was started on a special port.

3. DM3 Diagnostic Utility Descriptions

This section gives a description and parameters for each utility. The utilities are listed alphabetically. A brief description of each utility, along with a list of the operating systems on which it runs, is provided in Chapter 1, “Overview”.

The DM3 Diagnostic utilities include the following:

- Alarms
- Audio Controll
- CallInfo
- CAS Signal Editor
- DebugView
- Digit Detector
- DM3Insight
- Dm3KDebug
- Dm3post
- Dm3StdErr
- Dm3Trace
- Getver
- ISDN Trace
- KernelVersion
- LineAdmin
- Listboards
- MercMon
- PDK Configuration Utility
- PDKManager Utility
- Phone
- QError
- STD Config
- StrmStat

- TSP Config
- TSP Monitor
- TSP Tracer

3.1. Alarms

Alarms is used for sending and monitoring the alarm states on a T-1 or E-1 line. However, if you are already using *LineAdmin* to put lines into service, you may not need to use *Alarms* because *LineAdmin* displays much of the same information. Overall, *LineAdmin* is a more useful tool and should be used instead of *Alarms* in most cases. See Section 3.15, “LineAdmin”, on page 61.

Operating Systems: Windows, Linux

Command Line: `alarms [parameter_list]`

The *Alarms* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (required)

Example: This example monitors the alarm states on line 1 of board 0:

```
alarms -board 0 -line 1
```

Display: Figure 1 shows the *Alarms* display.

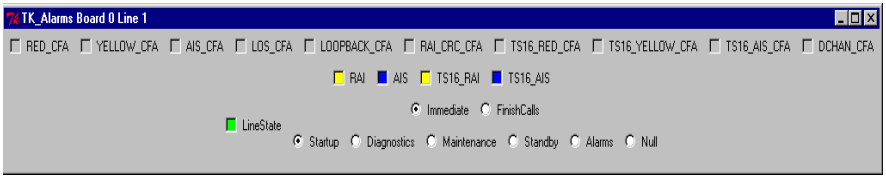

Figure 1. Alarms Display

3.2. Audio Control

Audio Control demonstrates the use of the Player and Recorder components. It provides control of the Player and Recorder resources, including speed and volume control. It also supports remote audio monitoring.

Operating Systems: Windows, Linux

Command Line: `audio [parameter_list]`

The *Audio Control* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-chan <*n*>	Channel number (optional, default is 1)

Example: This example runs the *Audio Control* utility on board 0, line 1, channel 1:

```
audio -board 0 -line 1 -channel 1
```

Display: Figure 2 shows the *Audio Control* display.

Figure 2. Audio Control Display

3.3. CallInfo

CallInfo detects call information using the Telephony Service Provider (TSP) resource. *Phone* also detects and displays call information, but *CallInfo* offers single call monitoring and more detail. *CallInfo* lets you specify the call-related events you want to trace. You can choose from the following:

- **CallInfoSet** - TSP-related call messages
- **IE Set** - Information elements of ISDN-related messages (focuses on small pieces of information)
- **ISDNMsgSet** - ISDN-related messages.

Operating Systems: Windows, Linux

3.3.1. Options

Command Line: `callinfo [parameter_list]`

The *CallInfo* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-chan <*n*>	Channel number (optional, default is 1)

Example: This example runs the *CalIInfo* utility on board 0, line 1, channel 1:

```
callinfo -board 0 -line 1 -chan 1
```

3.3.2. Guidelines

To use *CallInfo*, specify the call-related events you want to trace as follows:

1. Click the **Action** menu on the *CallInfo* display (Figure 3). Highlight **Select Ids** and select the group of call-related events you want to trace (**CallInfoSet**, **IE Set** or **ISDNMsgSet**).

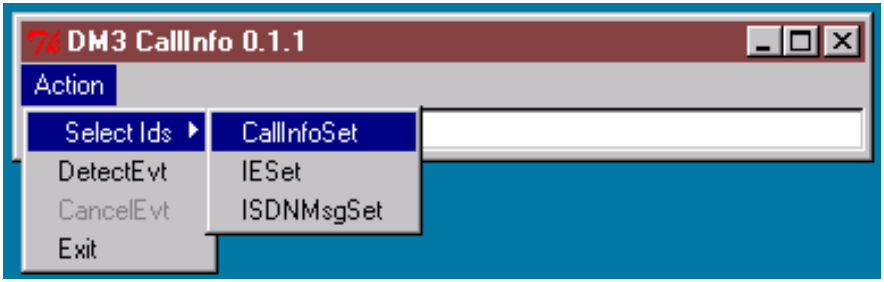

Figure 3. CallInfo Action Menu

2. A window of events specific to the event group you selected (**CallInfoSet**, **IE Set** or **ISDNMsgSet**) will appear. Figure 4 shows the window that will appear if you choose **ISDNMsgSet**. Select the events you want to trace by clicking on them.

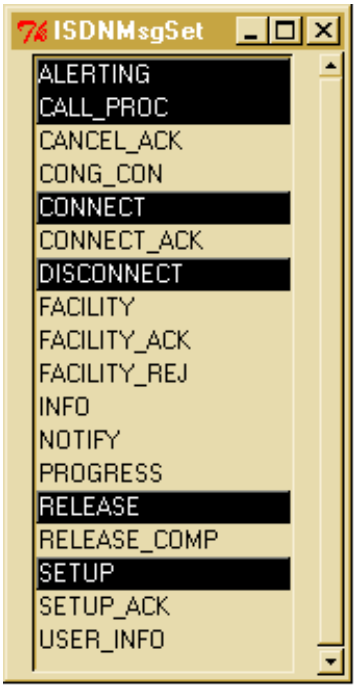

Figure 4. ISDNMsgSet Event Selection

3. After you select events, click the **Action** menu and choose **DetectEvt**. The utility will start tracing the events you selected. Figure 5 shows *CallInfo* tracing sample events. As new call information comes in, it will write over the old call information on the *CallInfo* display. A separate console window will open that tracks messages coming in and shows the message sequence.

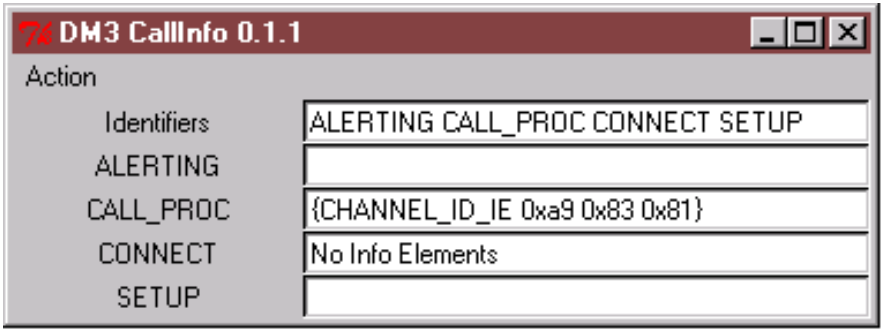

Figure 5. CallInfo Tracing Sample Events

4. If you want to stop tracing the events you selected and select other events to trace, click the **Action** menu and select **CancelEvt**. Then follow steps 1 through 3 again.

5. To exit the *CallInfo* tool completely, select **Action > Exit** or close the window.

3.4. CAS Signal Editor

The *CAS Signal Editor* allows you to dynamically view and modify CAS signal identification parameters (transitions, pulses, trains, or sequences) so you can test them before changing the *.config* file.

Signal identification parameters are defined by the *.config* and *.fcd* files, and downloaded to the board. To modify the parameters without the *CAS Signal Editor* utility, you must modify the signal definitions contained in the *.config* file, generate a new *.fcd* file, and then re-download the file to the board (this process is described in the *DM3 Configuration File Reference*). Using the *CAS Signal Editor* utility, you can retrieve the current signal identification parameters and reconfigure them at runtime without downloading to the board.

Operating Systems: Windows, Linux

3.4.1. Options

Command Line: `signaleditor [parameter_list]`

The *CAS Signal Editor* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-signal <*n*>	Signal ID (optional).

Example: This example runs the *CAS Signal Editor* utility on board 0:

```
signaleditor -board 0
```

3.4.2. Guidelines

Once you start the utility, a window will open in which you can choose the signal that you want to edit (Figure 6). For signal IDs, refer to the *DM3 Configuration File Reference*. You can edit a signal and check the results as follows:

1. In the SignalId field of the *CAS Signal Editor* display (Figure 6), enter the ID of the signal you want to edit.

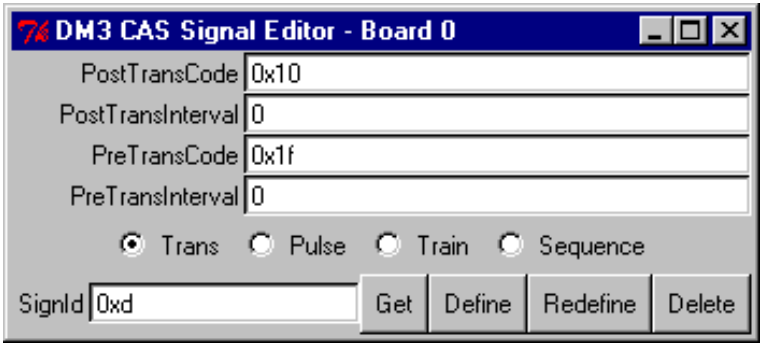

Figure 6. CAS Signal Editor Display

2. Select the appropriate category (Trans, Pulse, Train, or Sequence).

3. Click **Get**. The display will show the signal information you requested.

4. Edit the signal information as desired.

5. Click **Redefine** to apply the update.

6. If you wish, use *Phone* and *TSP Monitor* to check the changes you made to the signal.

The *CAS Signal Editor* also allows you to define a new signal (fill in the fields and click **Define**) and delete a signal (use **Get** to populate the display with the signal you want to delete and click **Delete**).

3.5. DebugView

DebugView[1] is an application that lets you monitor debug output on your local system, or any computer on the network that you can reach via TCP/IP. It can display both kernel-mode and Win32 debug output generated by standard debug print APIs, so you don't need a debugger to catch the debug output your applications or device drivers generate, and you don't need to modify your applications or drivers to use non-Windows debug functions in order to view its debug output.

Operating System: Windows

To download this freeware and get full instructions for using it, go to *http://www.sysinternals.com/ntw2k/freeware/debugview.shtml*.

3.6. Digit Detector

The *Digit Detector* demonstrates the use of the Tone Generator and Signal Detector components. It provides the ability to detect digits at the local end of a channel connection. To use *Digit Detector*, you need a physical connection. For example,

1. Copyright © 1998-2001 Mark Russinovich

two trunks can be looped or you might use a connection between end users. The *Phone* utility can be used at one end of the channel connection to generate the dialed digits. For details, see Section 3.20, “Phone”, on page 66.

Operating Systems: Windows, Linux

Command Line: `digitdetector [parameter_list]`

The *Digit Detector* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-chan <*n*>	Channel number (optional, default is 1)

Example: This example runs the *Digit Detector* utility on board 0:

```
digitdetector -board 0
```

Display: Figure 7 shows the *Digit Detector* and *Phone* displays. Note that digits (such as DTMF) are shown in alphanumeric form on the *Digit Detector* display.

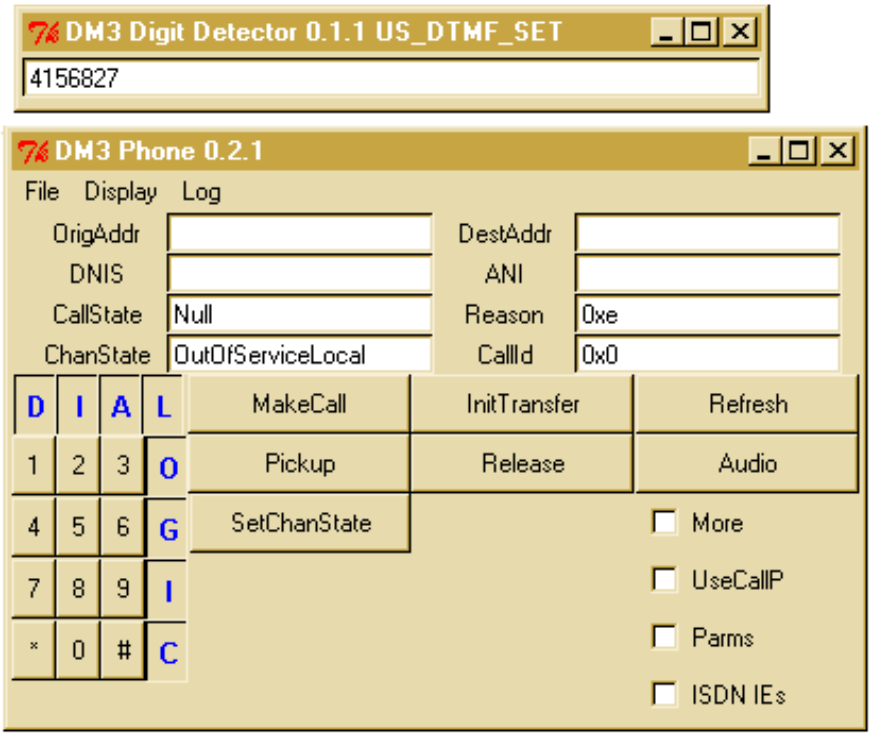

Figure 7. Digit Detector and Phone Displays

3.7. DM3Insight

DM3Insight is a tool designed to trace and decode message and stream traffic between host and DM3 boards. *DM3Insight* consists of a driver and GUI application. The driver is responsible for capturing message/stream traffic and decoding it. The GUI is used to set various trace options.

Operating System: Windows

DM3Insight can be useful in

- Debugging and/or understanding applications, libraries, drivers, kernel, and firmware by capturing and analyzing the trace output. This can be done either locally or remotely.
- Finding the turnaround time for messages.
- Viewing the messages that are encountered only between the driver and boards.
- Eliminating orphan messages and streams.

3.7.1. Options

DM3Insight can run in one of the following modes:

- **Local Client** - Directly communicates with drivers on the same system.
- **Server** (Application / Service) - Accepts trace/dump commands from a remote system and forwards them to the driver.
- **Remote Client** - Communicates with a Server from a remote system to send trace/dump commands.

Instructions for using these modes are provided in the following sections:

- Local Mode
- Remote Mode

Local Mode

This is the simplest mode of operation where the application is running on the host system (the system that has DM3 boards). No command line parameters are required.

1. Execute the application with the following command: `DM3Insight`. The *DM3Insight* GUI appears (Figure 8).

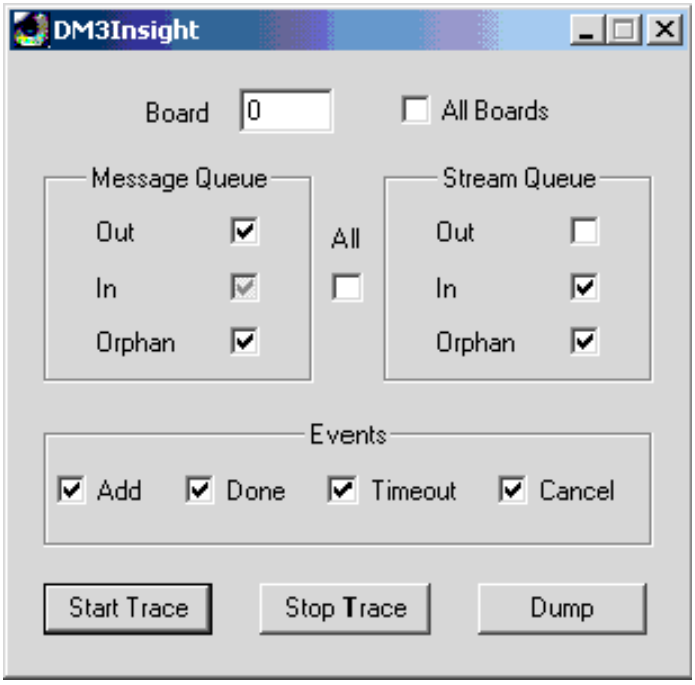

Figure 8. DM3Insight GUI

2. Select the options you want:
   - **Board** - choose a specific board or all boards
   - **Message Queue** - choose from Message Out, In, and Orphan
   - **Stream Queue** - choose from Stream Out, In, and Orphan
   - **Events** - Add, Done, Timeout, and Cancel

3. To start capturing message / stream traffic, click the **Start Trace** button.

4. To view the data, run *DebugView* (see Section 3.5, “DebugView”, on page 19).

5. To dump the queues, click the **Dump** button.

Remote Mode

The application is run in server mode on the host system if it requires having a GUI running on a remote system. Some command line parameters are required. These are explained as required in the procedure, but summarized here:

Parameter	Description
-h or -?	Show all command line parameters
-cr	Run as Server in application mode
-ci	Install and run as Server in service mode
-cu	Uninstall Server service
-p<*Port*>	TCP/UDP port to use for Remote operation. Default is 5000
-t	Enable printing of driver / application error messages

(If you don't specify the -cr, -ci, or -cu parameter, the application runs in Local Mode.)

1. From the host system, execute the application with the following command line: `DM3Insight -cr`

   The **DM3Insight: Remote** window (Figure 9) appears.

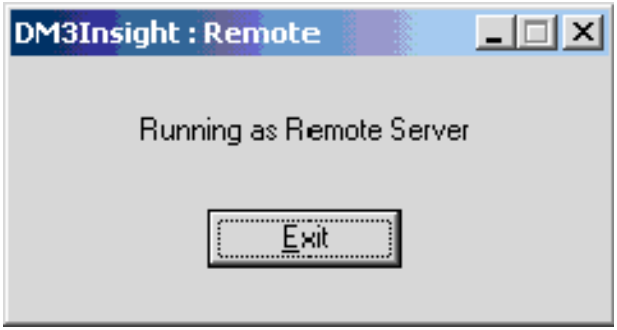

Figure 9. DM3Insight: Remote

DM3Insight is now running in Server mode and waiting for an incoming connection

2. Now run *DM3Insight* on a remote system without any command line parameters.

   2.a. From the **System Menu** (click on the **System Menu** icon or press **Alt+Space**), select the **Connect** option. The **DM3Insight: Connect** window appears (Figure 10).

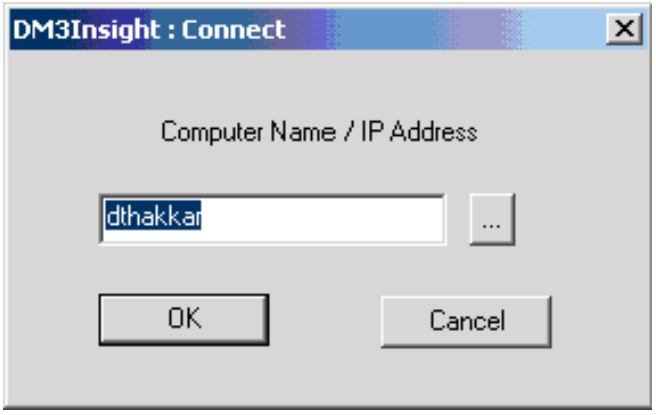

Figure 10. DM3Insight: Connect

   2.b. Fill in the name or IP address of the host system (see Figure 10). If you need to browse for the host system, click the button with the three dots to the right of the place in which you type text (Figure 10). The **Browse for Computer** screen appears (Figure 11).

   2.c. Browse to find the name or IP address of the host system.

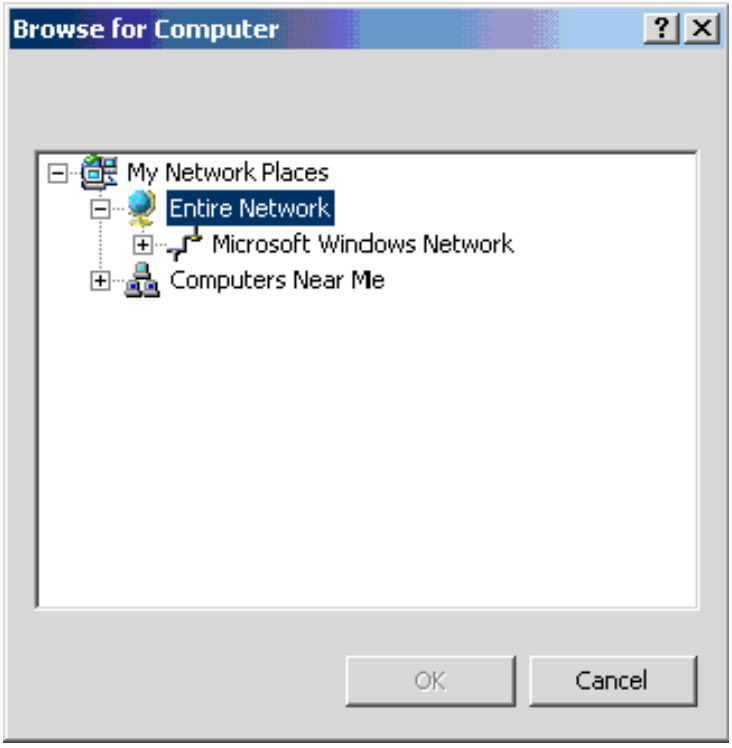

Figure 11. Browse for Computer

If the connection is successful, you'll see the name of host system on the title bar of the GUI (Figure 12).

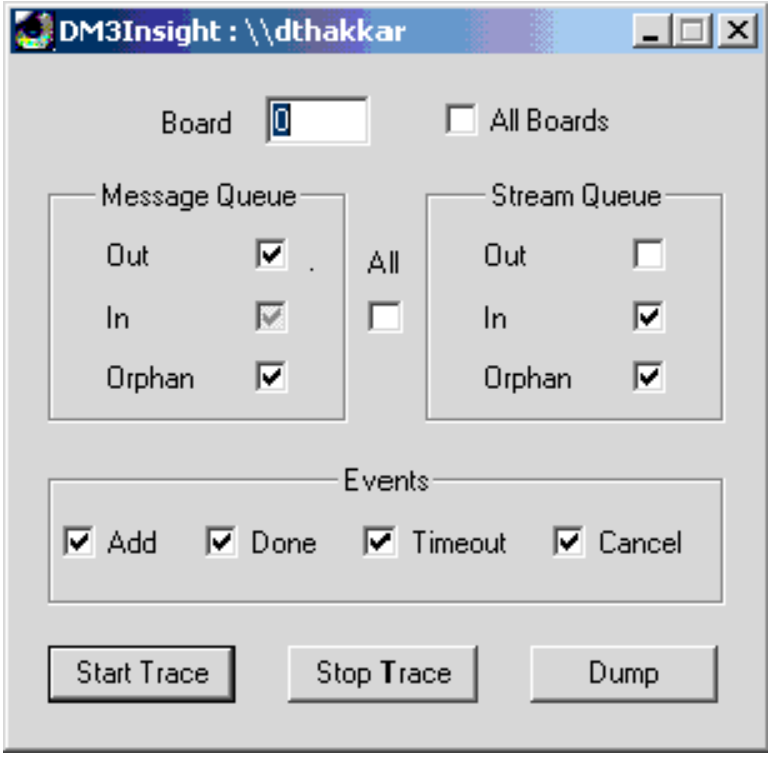

Figure 12. DM3Insight GUI - Host System

3. Now you can control tracing or dump the queues just like in Local Mode, but in this case the GUI is running in Remote Client Mode. Select the options you want.

4. From the Remote system, connect *DebugView* to the host system. You may need to run *DebugView* in client mode on the host system.

5. Once you have made a successful connection, you can run *DM3Insight* in server mode as a service. Run *DM3Insight* on the host system with the following command line parameters:

   ```
   DM3Insight -ci
   ```

After doing this, you will not have to bother running *DM3Insight* on the host system as Server again, even if you reboot the system.

6. To uninstall *DM3Insight* service use the following parameter:
   ```
   DM3Insight -cu
   ```

3.7.2. Filtering the Output

Once you have captured the data for all the channels, if you need to filter out data for some particular channel or extract particular messages or stream traffic, use the **Filter** option as described in this procedure:

1. Select the **Filter** option from the System Menu. The **DM3Insight Filter** screen appears (Figure 13).

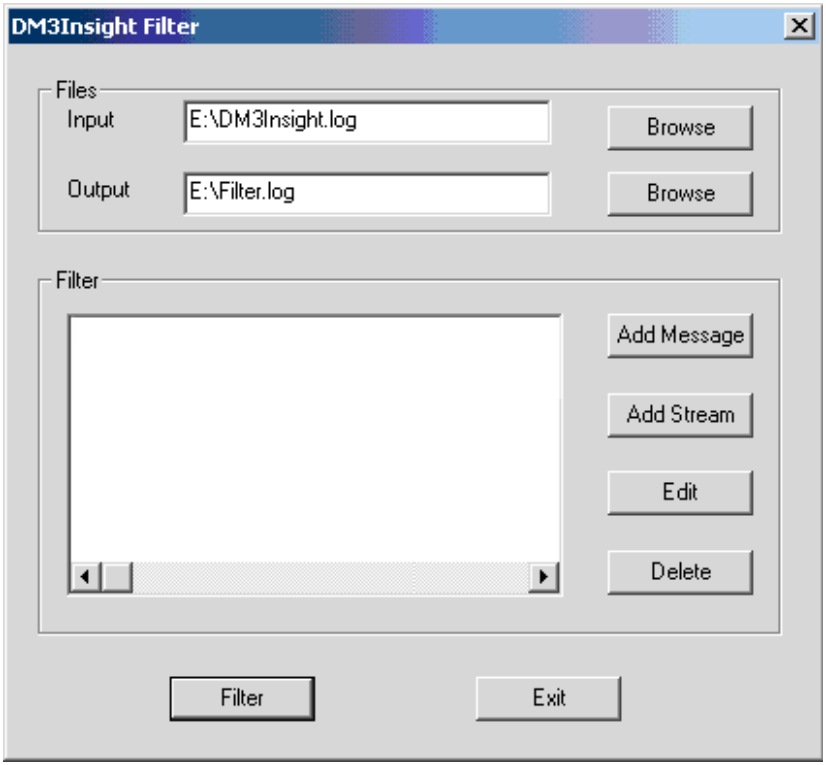

Figure 13. DM3Insight Filter

2. Under Files, specify the Input and Output files you want to use. Click on **Browse** to locate the file you want.

3. Click the **Add Message** button.

4. The **DM3Insight Filter: Message** screen appears (Figure 14).

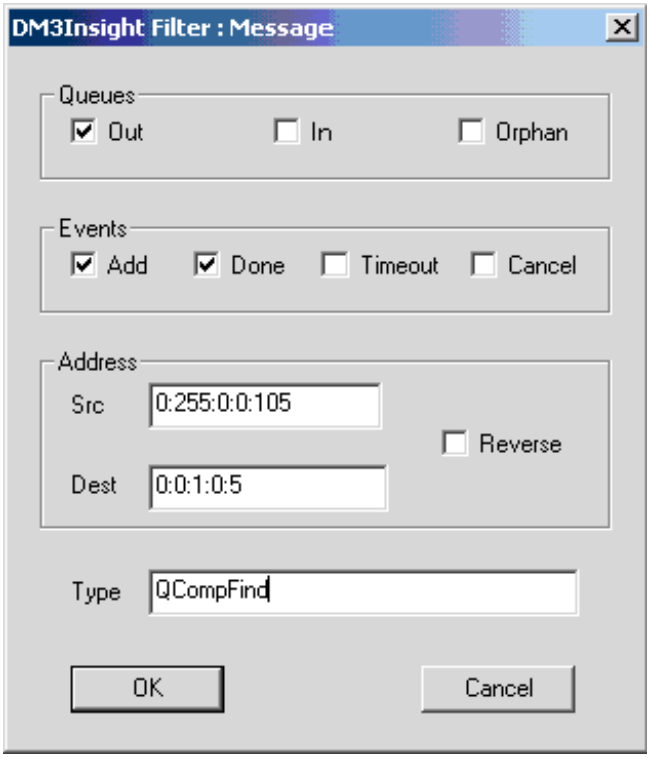

Figure 14. DM3Insight Filter: Message

5. To extract message events with a specific message type, enter the symbolic name of the message in the Type field.

6. Click **OK**. You'll return to the **DM3Insight Filter** screen (Figure 13).

7. Click the **Add Stream** button. The **DM3Insight Filter: Stream** screen appears (Figure 15).

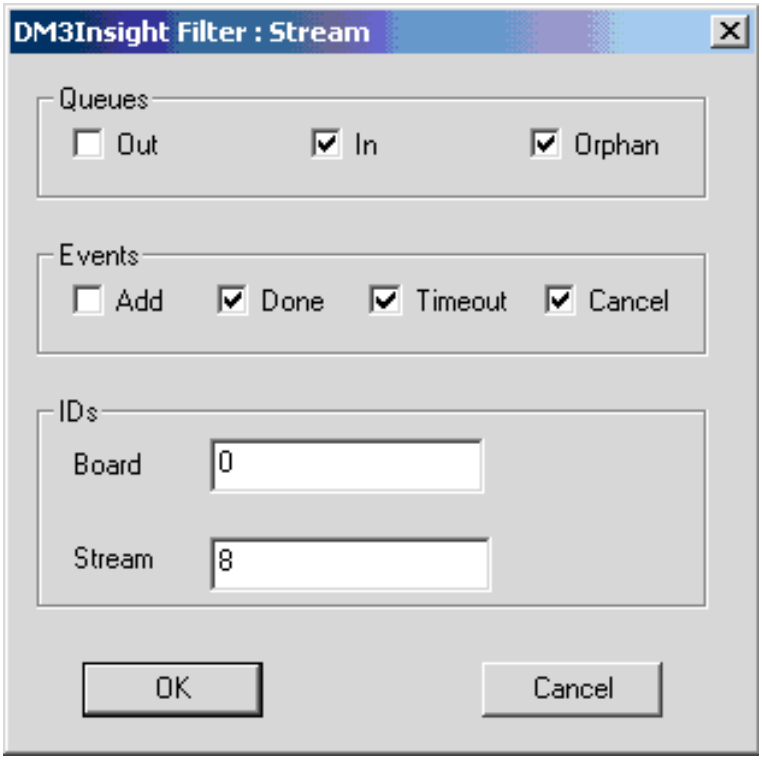

Figure 15. DM3Insight Filter: Stream

8. On this screen (Figure 15), specify the board number and stream ID of stream events you're interested in.

9. Click **OK**. The **DM3Insight Filter** screen (Figure 16) will show the selections you've made so far.

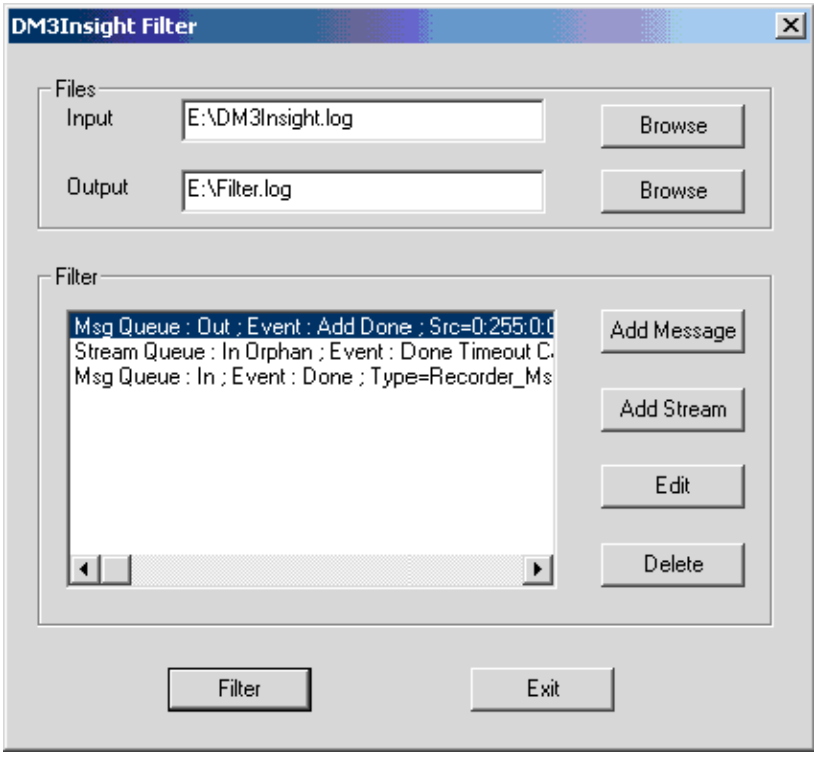

Figure 16. DM3Insight Filter - Message and Stream Filters Added

10. You can add up to 50 message or stream filters. In each filter you can select the queue type from Out, In and Orphan queue. The choice for events is Add, Done, Timeout, and Cancel.

    For message filters you can specify the Source/Destination address in *node:board:processor:component:instance* format. If you don't care whether these addresses appear in Source or Destination address fields, select Reverse.

11. Once you've added all the message and stream filters you want and selected input and output files, you can run the filters by clicking the **Filter** button.

3.7.3. Viewing the Hex Dump of Message Events

To view the hex dump of message events, do the following:

1. Select **Options** from the System Menu. **The DM3Insight: Options** screen appears (Figure 17).

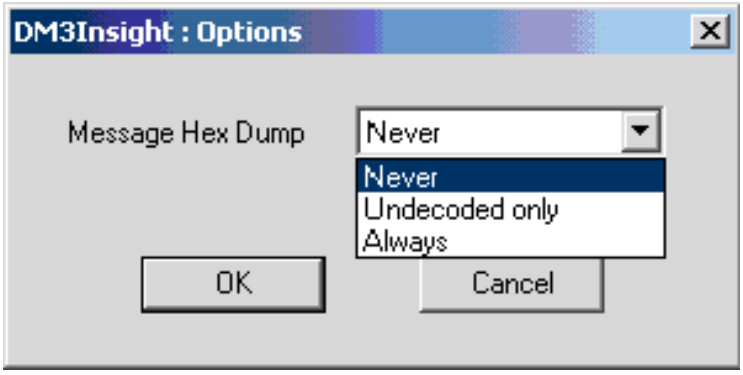

Figure 17. DM3Insight: Options

2. From the drop down list on this screen (Figure 17), select a Message Hex Dump option from the following:
   - **Never**: Never show any hex dump
   - **Undecoded only**: Show hex dump of message events that the *DM3Insight* driver could not decode.
   - **Always**: Always show the hex dump

3. Click **OK**.

3.7.4. Guidelines

This information provides the following guidelines for using *DM3Insight*:

- Troubleshooting
- Limiting Memory/CPU Usage

Troubleshooting

If things don't work:

- Make sure that *DM3InsightDrv.sys* is copied in the \WinNT\system32\drivers directory.
- Enable printing of error messages by running *DM3Insight* with *-t* parameters to get more detailed error messages.
- To capture traffic during the download process, select **All boards** instead of a specific board as the logical board ID may not be assigned.
- If there is no trace output in *DebugView*, make sure that **Capture Kernel** and **Capture Events** are selected in the **Capture** menu of *DebugView.*
- When using *DM3Insight* in remote mode, make sure that you can ping the host system. Make sure that the TCP/UDP port used by *DM3Insight* is not used by any other application using the *netstat* application.
- Always run the *DM3Insight* server on the host system before running the remote *DM3Insight* client.

Limiting Memory/CPU Usage

DebugView allocates memory to save the output from *DM3Insight*. If the amount of traffic is large, it is advisable to limit the history depth (number of lines buffered) and select the **Log to File** option.

If CPU usage is too high, consider limiting the queues or events. Minimizing the *DebugView* window or capturing traffic on remote system also reduces CPU usage.

3.8. Dm3KDebug

Dm3KDebug queries the DM3 Kernel for debug information.

Operating Systems: Windows, Linux

Command Line: Once invoked and connected to the board, *Dm3KDebug* displays a console window with a command prompt in which debugging parameters may be specified. The debug information is posted to the screen and to the designated output file.

```
dm3kdebug [parameter_list]
```

The *Dm3KDebug* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-d<*level*>	Do not modify. Leave at default value of 0.
-p<*n*>	Processor number (required)
-f <*file*>	Output file name (required to capture output in a file)
-h	Displays the program help screen (optional)
-v	Displays the program version (optional)

Example: This example displays a *Dm3KDebug* prompt:

```
dm3kdebug -b1 -p7
```

For a list of available debugging options, type `help` or see Table 2.

Table 2. Dm3KDebug Debugging Options

Option	Description
exit	Exits the *Dm3KDebug* utility
exitlist	Prints host exit notification list
help	Displays this list of available debugging options
notifylist	Prints clients that are notified on exit notification
pools	Prints local pool information

Table 2. Dm3KDebug Debugging Options (Continued)

Option	Description
port <comp> <instance> <portType> <portDirection> <portInstance>	Prints port information about the component/instance: • <portType>: MMA use 1, SCbus use 2, NIC use 3 • <portDirection>: IN use 1, OUT use 2, ANY use 0 • <portInstance>: unique number assigned; 255 is wildcard
registry <startProc> <startComp> <startInst> <endProc> <endComp> <endInst>	Prints out the registry database. Arguments give the start and stop points for each search.
streams <stop point>	Prints stream information with <stop point> being the last stream to print
quit	Quits the *Dm3KDebug* utility
cluster <instance>	Prints cluster information about the cluster instance number specified

3.9. Dm3post

The *dm3post* diagnostic tool, sometimes referred to as "POST On-Demand," can perform diagnostics on a stopped board at any time to detect and isolate possible hardware faults. This can enable you to locate and diagnose faulty boards.

Operating Systems: Windows, Linux

3.9.1. Options

This information describes the two ways to run *dm3post*:

- Running dm3post from the Command Line
- Running dm3post via SNMP

Running *dm3post* from the Command Line

The run command for *dm3post* is as follows:

```
dm3post [parameter_list]
```

The *dm3post* tool uses the command-line parameters listed below. The user may choose to run diagnostics, display the help screen, or display the program version, but only one request is supported per invocation.

Parameter	Description
-s<*n*>	Slot number (required) where n = 0..31
-b<*n*>	Bus number (optional if there is only one bus OR if slot number is unique when there is more than one bus) where n = 0..6 for WINDOWS and n = 0..7 for Linux
-h	Displays the program help (optional). No diagnostics will be run if this parameter is used.
-v	Displays the program version (optional). No diagnostics will be run if this parameter is used.

Example: This example runs dm3post on slot 17, bus 0:

```
dm3post -s17 -b0
```

You will get this response:

```
Do you wish to continue (y/n)?
```

If you answer **Y**, the following will be printed to the screen:

```
dm3post processing…
```

The success or error message will be printed.

Running *dm3post* via SNMP

To run *dm3post* via Simple Network Management Protocol (SNMP), perform the following steps:

1. Stop the board to be diagnosed.

2. Using an SNMP MIB browser, locate and highlight the following object: **dlgHiIdentAdminStatus.**

3. Enter a **0** in the **Mib Instance** window and a **3** in the **SNMP Set Value** window.

4. Verify that the board has stopped.

5. Run *dm3post*.

6. Using an SNMP MIB browser, locate and highlight the following object: **dlgHiIdentAdminStatus.**

7. Enter a **0** in the **Mib Instance** window and a **5** in the **SNMP Set Value** window.

8. POST diagnostics will start to run. They should take about 60 to 90 seconds to complete.

Upon completion, the **dlgHiIdentAdminStatus** variable will transition from **Diagnose** to **Stopped** and a state transition trap will be generated. A text description of the diagnostic result will be stored in the **dlgHiIdentErrorMessage** variable associated with the board.

3.9.2. Guidelines

The DM3 board must be in a stopped state before *dm3post* can be run. *Dm3post* will reset the specified board forcing the Control Processor (CP) POST diagnostics to run. *Dm3post* will then retrieve the POST results from SRAM and provide a PASS/FAIL indication to the user. The board will remain in a stopped state

following completion of POST diagnostics and the user will be required to restart the board.

3.9.3. Dm3post-Defined Diagnostic Codes

In addition to the POST diagnostic codes retrieved from SRAM, it is necessary to define diagnostic codes for all possible error conditions that can occur during *dm3post's* execution. Table 3 lists the *dm3post*-defined diagnostic codes and associated error conditions. The legend for Table 3 is as follows:

- **Diag Code 1** - the diagnostic code returned from SRAM diagnostic location 1 or the *dm3post*-defined diagnostic code.
- **Diag Code 2** - the diagnostic code returned from SRAM diagnostic location 2 or the *dm3post*-defined diagnostic code
- **Windows** - X indicates that the message is supported on the Windows platform
- **Linux** - X indicates that the message is supported on the Linux platform
- **Message / Action** - the message displayed and the associated action to be taken

Table 3. Dm3post Output Messages

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0x00 OR 0x01	Test # in Progress	X	X	Message	ERROR: Power On Self Test (test name) timed out for board in slot nn, bus nn. Diagnostic codes: 0xnn 0xnn

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
				Action	Rerun *dm3post*. If the problem persists, provide the error message and diagnostic codes to Dialogic customer support.
0x03	0xFC	X	X	Message	SUCCESS: Power On Self Test passed for board in slot nn, bus nn. Diagnostic Codes: 0x03 0xFC
				Action	None
Test# which Failed	Test# which Failed	X	X	Message	ERROR: Power On Self Test (test name) failed for board in slot nn, bus nn. Diagnostic Codes: 0xnn 0xnn
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
Invalid combination of diagnostic codes		X	X	Message	ERROR: Invalid results from Power On Self Test for board in slot nn, bus nn. Diagnostic Codes: 0xnn 0xnn

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x01	X	X	Message	ERROR: Invalid entry. Valid range for slot number is [0 - 31]. Diagnostic Codes: 0xFF 0x01
				Action	Rerun *dm3post* and provide a valid slot number.
0xFF	0x02	X	X	Message	ERROR: No DM3 board found with slot nn and bus nn. Diagnostic Codes: 0xFF 0x02
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x03	X	X	Message	ERROR: Slot number must be entered. Diagnostic Codes: 0xFF 0x03
				Action	Rerun *dm3post* and provide a valid slot number.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x04	X	X	Message	ERROR: Only one slot number may be entered. Diagnostic Codes: 0xFF 0x04
				Action	Rerun *dm3post* and provide a valid slot number.
0xFF	0x05	X	X	Message	ERROR: Only one bus number may be entered. Diagnostic Codes: 0xFF 0x05
				Action	Rerun *dm3post* and provide a valid bus number.
0xFF	0x06	X	X	Message	ERROR: Ambiguous slot number. Must enter bus number to uniquely identify board. Diagnostic Codes: 0xFF 0x06
				Action	Rerun *dm3post* and provide valid slot and bus numbers.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x07	X	X	Message	ERROR: qMsgVarFieldPut() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x07
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x08	X	X	Message	ERROR: Board must be stopped/reset before diagnostics can run. Diagnostic Codes: 0xFF 0x08
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x30	X		Message	ERROR: mntEnumMpathDevice() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x30
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x31	X		Message	ERROR: CreateFile() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x31
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x32	X		Message	ERROR: mntAllocateMMB() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x32
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x33	X		Message	ERROR: mntSendPostMessage() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x33
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x34	X		Message	ERROR: mntResetBoard() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x34
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x35	X		Message	ERROR: NCM_GetValueEx() - <error msg>. Diagnostic Codes: 0xFF 0x35
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x36	X		Message	ERROR: NCM_GetInstalledDevices() - <error msg>. Diagnostic Codes: 0xFF 0x36
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x37	X		Message	ERROR: mntGetPOSTLocationContent() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x37
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x38	X		Message	ERROR: Invalid entry. Valid range for bus number is [0-6]. Diagnostic Codes: 0xFF 0x38
				Action	Rerun *dm3post* and provide a valid bus number.
0xFF	0x39	X		Message	ERROR: DeviceIoControl() failed. Diagnostic Codes: 0xFF 0x39
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	**Diag Code 2**	**Windows**	**Linux**	**Message / Action**	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x50		X	Message	ERROR: Invalid entry. Valid range for bus number is [0 - 7]. Diagnostic Codes: 0xFF 0x50
				Action	Rerun *dm3post* and provide a valid bus number.
0xFF	0x51		X	Message	ERROR: qDrvSetInterface() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x51
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x52		X	Message	ERROR: qQueueOpen() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x52
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x53		X	Message	ERROR: qQueueBind() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x53
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x54		X	Message	ERROR: qDrvBrdMap() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x54
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x55		X	Message	ERROR: qDrvReportBoards() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x55
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x56		X	Message	ERROR: qDrvReportBrdStates() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x56
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x57		X	Message	ERROR: qDrvBrdShutdown() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x57
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x58		X	Message	ERROR: qDrvBrdGetDiagState() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x58
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x59		X	Message	ERROR: qDrvBrdStart() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x59
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x5A		X	Message	ERROR: qDrvProtStart() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x5A
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x5B		X	Message	ERROR: Unable to retrieve board state for board in slot nn, bus nn. Diagnostic Codes: 0xFF 0x5B
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x5C		X	Message	ERROR: qMsgAllocate() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x5C
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x5D		X	Message	ERROR: qMsgWrite() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x5D
				Action	Provide the error message and diagnostic codes to Dialogic customer support.
0xFF	0x5E		X	Message	ERROR: qDrvBrdConfig() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x5E
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

Table 3. Dm3post Output Messages (Continued)

Diag Code 1	Diag Code 2	Windows	Linux	Message / Action	
Diagnostic Codes returned by POST				Error messages returned in diagMsg to calling routine	
0xFF	0x5F		X	Message	ERROR: qMsgRead() failed. Error code = nnn. Diagnostic Codes: 0xFF 0x5F
				Action	Provide the error message and diagnostic codes to Dialogic customer support.

3.9.4. Power On Self Test (POST)

This section provides additional information on Power On Self Test (POST), the series of diagnostic tests used to verify that the DM3 board components are working properly. As described above, the *dm3post* diagnostic tool can run POST "on demand." But otherwise, the complete POST suite of tests runs automatically when a board detects that a power on condition has occurred. A power on condition occurs only when the host is started from a powered down state (cold boot). If the host resets the board (warm boot), then only a subset of the tests run.

POST status is displayed differently for digital network interface (DNI) boards and analog network interface boards. The following sections describe how to determine POST results for each type of board, and what to do if a POST failure occurs:

- Determining POST Status on Analog Network Interface Boards
- Determining POST Status on Digital Network Interface Boards

NOTE: Some of the following information is specific to QuadSpan products. Other DM3 products may provide a different display. POST display information for other products will be provided in a future issue of this document.

Determining POST Status on Analog Network Interface Boards

To determine POST status on analog network interface boards, follow these steps:

1. Locate the set of three LEDs located on the faceplate of the DM3 board. These LEDs are shown in Figure 18.

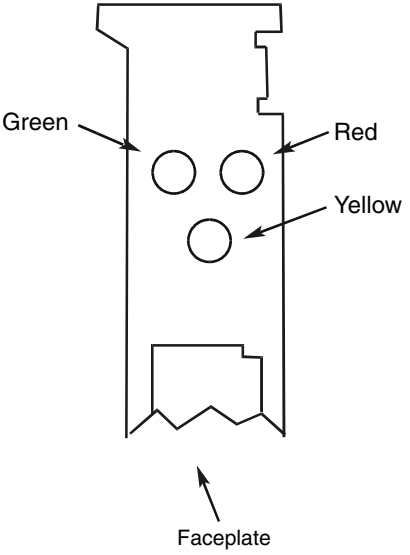

Figure 18. Analog Network Interface Board LEDs

2. Observe the LEDs during POST execution. They flash on and off as each test is run.
   - If POST passes all diagnostic tests, the LEDs turn off within 90 seconds of power up indicating POST completed successfully.
   - If POST fails a diagnostic test, the yellow LED flashes repeatedly after power up indicating POST failed.

Determining POST Status on Digital Network Interface Boards

To determine POST status on digital network interface boards, follow these steps:

1. Locate the two banks of LEDs on the DM3 board. On a cPCI board, the LEDs are located on the faceplate of the DM3 board as shown in Figure 19.

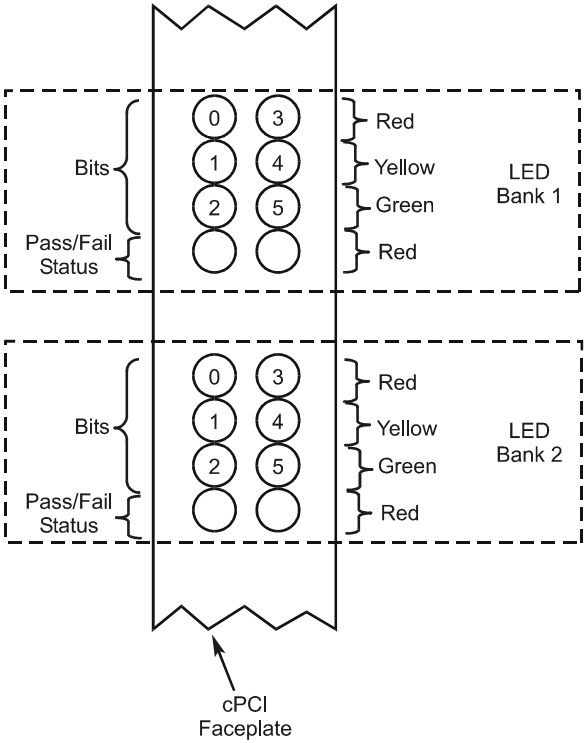

Figure 19. DNI Board LEDs (cPCI)

On a PCI board, the LEDs are located on the outer edge of the DM3 board as shown in Figure 20.

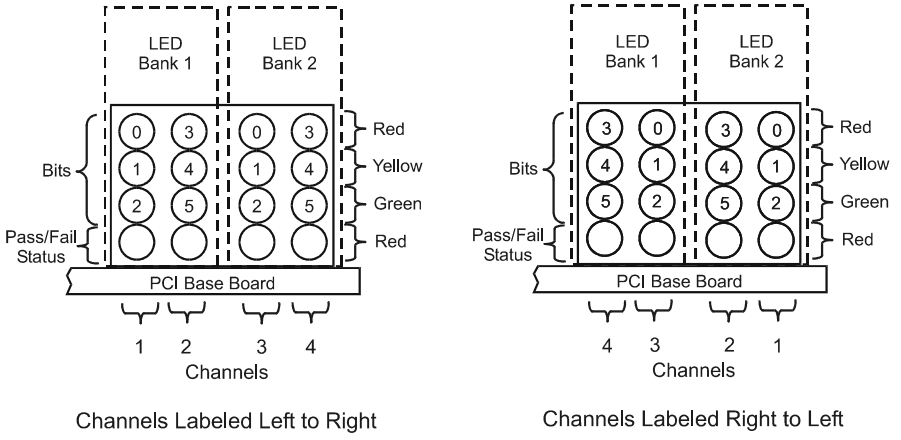

Figure 20. DNI Board LEDs (PCI)

2. Observe the LEDs during POST execution. A series of LEDs in both bank 1 and bank 2 flash on and off as each test is run.
   - If POST passes all diagnostic tests, the LEDs in both banks turn off indicating POST completed successfully.
   - If POST fails a diagnostic test, POST terminates immediately and displays the failing results in both banks of LEDs as follows:
     - a combination of LEDs in the first three rows of each bank remain on
     - the bottom row of LEDs in each bank flash repeatedly indicating a failure has occurred
     - both banks of LEDs display the same LED pattern.

     For example, Figure 21 shows a DNI cPCI board POST failure. The last row of LEDs would be flashing, indicating a failure.

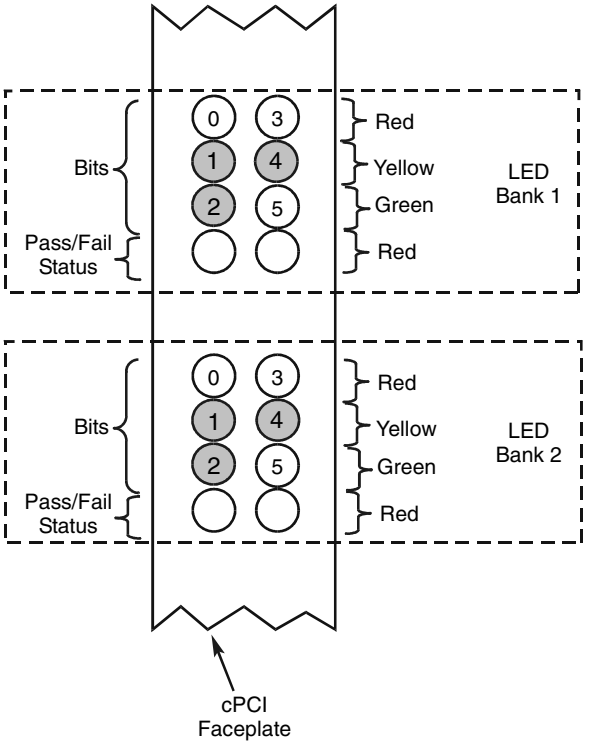

Figure 21. DNI cPCI Board POST Failure

3.10. Dm3StdErr

Dm3StdErr acts as a virtual serial port (terminal emulation) session to the DM3 board. The DM3 Kernel and resources can be enabled to send **qPrintf( )** statements to both the serial port and the host. This program polls the board and posts the **qPrintf( )** statements from the resources and DM3 Kernel to the screen and designated output file.

Operating Systems: Windows, Linux

3.10.1. Options

Command Line: `dm3stderr [parameter_list]`

The *Dm3StdErr* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	Board number (required, default is 0). Use the *Listboards* utility to obtain the board number.
-d<*level*>	Do not modify. Leave at default value of 0.
-f <*file*>	Output file name (required to save output in a file)
-s<*n*>	Stream number. Do not modify. Leave at default value of 1.
-h	Displays the program help screen (optional)
-v	Displays the program version (optional)

Example: This example receives the **qPrintf( ) d**ata from board 1, displays it to the screen, and saves it to file *output.txt*:

```
dm3stderr -b1 -f output.txt
```

3.10.2. Guidelines

Dm3StdErr and *Dm3Trace* both act as a virtual serial port (terminal emulation) session to the DM3 board, but they differ as follows:

- If the firmware prints out a string using the **qPrintf()** function call, the output string goes to the **com port/dm3Stderr** window.
- If the firmware prints out a string using the **qTrace()** function call, the output string goes to *Dm3Trace*.
- *Dm3StdErr* requires a board number, but *Dm3Trace* requires both a board number and a processor number
- *Dm3StdErr* posts **qPrintf()** from all the processors on a board, but *Dm3Trace* posts **qTrace()** from the indicated processor

- *Dm3Trace* allows for an optional argument: trace level, which is a bit mask allowing up to 24 simultaneous active trace levels, whereas *Dm3Trace* will filter out the output of **qTrace()** that is not in the current trace level.

3.11. Dm3Trace

Dm3Trace acts as a virtual serial port (terminal emulation) session to the DM3 board. This program continuously prints out the **qTrace** statements included in the DM3 Kernel with the **qTraceLevel( )** function. This program polls the board and posts the **qTrace** statements from the resources and DM3 Kernel to the screen and designated output file.

For guidelines on the differences between *Dm3Trace* and *Dm3StdErr*, see Section 3.10.2, “Guidelines”, on page 57.

Operating Systems: Windows, Linux

Command Line: `dm3trace [parameter_list]`

The *Dm3Trace* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-d<*level*>	Do not modify. Leave at default value of 0.
-f <*file*>	Output file name (required to save output in a file)
-p<*n*>	Processor number (required, lowest number is 1))
-t<*n*>	Reserved value. Leave at default value of 0.
-h	Displays the program help screen (optional)
-v	Displays the program version (optional)

Example: This example receives the **qTrace** data from board 1 and displays it to the screen:

```
dm3trace -b1 -p1
```

3.12. Getver

Getver displays the version of a DM3 binary file. It scans the binary for the standard DM3 Dialogic version string and prints it to the screen.

Operating Systems: Windows, Linux

Command Line: `getver <file>`

Examples:

This Windows example scans *fcdgen.exe* and prints the version string:

```
getver "C:\Program Files\Dialogic\bin\fcdgen.exe"
```

This example scans the Linux QVS_T1.MLM and prints out the version string of each component in it:

```
getver ../data/qvs_t1.mlm
```

3.13. ISDN Trace

ISDNtrace provides the ability to track Layer 3 (Q.931) messages on the ISDN D-channel. *ISDNtrace* prints the messages on the screen in real time. This trace information also can be captured into a file.

Operating Systems: Windows, Linux

Command Line: `isdntrace [parameter_list]`

The *ISDNtrace* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	The number of the board on which you want to run the utility (required). Use the *Listboards* utility to obtain the board number.
-d<*n*>	The D-channel number (trunk number) on the above board (required)
-f<*file*>	The filename into which the trace can be captured (required to save the trace in a file)

Example: This example runs *ISDNtrace* on board 0, D-channel (trunk) number 1 and captures the trace in a file named *logfile.txt*:

```
isdntrace -b0 -d1 -file logfile.txt
```

3.14. KernelVersion

KernelVersion queries the DM3 Kernel running on a particular processor for its version number. The `-l` parameter is useful to "ping" the board to generate message traffic. *KernelVersion* is also useful to verify that the DM3 Kernel is running.

The current version of *KernelVersion* requires that you enter board numbers starting at 0. For example, the first board installed in your system is board 0, the second is board 1, and so on.

Operating Systems: Windows, Linux

Command Line: `kernelver [parameter_list]`

The *KernelVersion* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-d<*level*>	Do not modify. Leave at default value of 0.
-p<*n*>	Processor number (required)
-l<*n*>	Loop number. Number of times the program will retrieve the version (optional)
-h	Displays the program help screen (optional)
-v	Displays the program version (optional)

Example: This example gets the version of the DM3 Kernel running on processor 2 of board 0:

```
kernelver -b0 -p2
```

3.15. LineAdmin

LineAdmin puts lines into service so you can run many of the other diagnostic utilities. *LineAdmin,* like *Alarms,* is used for sending and monitoring the alarm states on a T-1 or E-1 line but *LineAdmin* is recommended as a more useful tool.

A flexible logging feature is available that includes the ability to log the status of the DM3 trunks and alarm conditions. For Linux systems, this utility also has the ability to re-initialize itself after every board download (based on a reset command from the user).

Operating Systems: Windows, Linux

Command Line: `lineadmin [parameter_list]`

The *LineAdmin* utility uses the following command line parameters:

Parameters	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-lines {*n n*+ …}	Line numbers. This parameter is used when more than 1 line is monitored (optional, default is {1 2 3 4})
-advanced <*n*>	The presence or absence of the following alarms on the line: AIS, CRC, and D-Channel
-ts16 <*n*>	The presence or absence of the following alarms on the Timeslot 16 of an E-1 trunk: Red Alarm, Yellow Alarm, and AIS
-log <*filename*>	Output log file (optional)

Example: This example runs the *LineAdmin* utility on board 1, lines 1, 2, 3, and 4:

```
lineadmin -board 1 -lines {1 2 3 4}
```

Display: Figure 22 shows the *LineAdmin* display. The display shows four trunks (a QuadSpan product). The alarm setting is on the left and the alarm indicators are on the right.

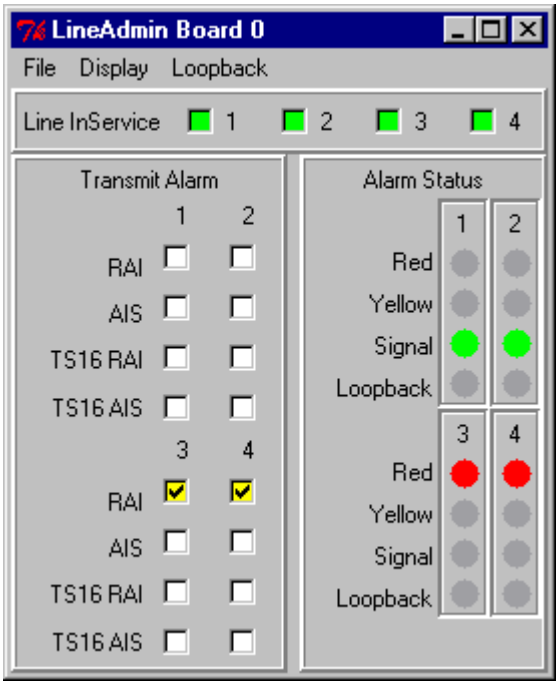

Figure 22. Line Admin Display

3.16. Listboards

Listboards prints information for DM3 board(s) present in the system and recognized by the device driver. *Listboards* prints complete information regarding the current status of the baseboard along with a list of attached digital network interface and processing daughterboards (if any).

Operating Systems: Windows, Linux

3.16.1. Options

Command Line: `listboards [parameter_list]`

The *Listboards* utility uses the following command line parameters:

Parameters	Description
-b <*n*>	Board number (optional). If no board number is specified, the status of all boards in the system will be given.
-c <*slotnumber*>	Retrieve config ROM from specified slot (optional)
-d <*level*>	Application debug level (optional)
-h	Help (optional)
-i <*board number*>	Retrieve hardware information (optional)
-s <*slotnumber*>	Scan slot number to see if a board is present (optional)
-v	Version (optional)

Example: This example lists all the board attributes for board 0. If any daughterboards are present, their attributes are also listed:

```
listboards –b0
```

3.16.2. Guidelines

On Linux, *Listboards* provides a logical board number and a physical board number, which is the number of the slots in the chassis. On Windows, it only provides a logical board number. The *logical* board number should be used when running any of the DM3 diagnostic utilities that require a board number.

Table 4 explains how board numbers are assigned based on the operating system and expansion card slots used.

Table 4. Board Numbers

Operating System	Data Bus	Board Number Assignments
Windows	PCI	Board numbers are dynamically assigned. *Listboards* prints out the logical board number along with the board's serial number. Use the serial number to physically identify the board on the PCI chassis.
	cPCI	See Windows PCI.
Linux	PCI	The physical board number is equivalent to the slot number of the board present on the chassis. This slot number is configured by setting the rotary switch present on the board.
	cPCI	Not available

Listboards will fail if you specify a board number that does not exist on the system.

On Linux systems, if *no* board number is specified, *listboards* will return the status of *all* boards in the system. Figure 23 shows sample output from running

```
listboards -c3
```

which retrieves the config ROM for slot 3.

```
MDI Library   version: 4.00  Build: 2
DM3pp Library version: 1.00 Beta 6 Build: 03
Driver version       : 4.02  Build: 03

BrdNum CfgId  Type Bus Slot  PhysAddr  RamSize  Irq Vector  State
   3     0      P   0    1  ff000000    80000    1   0x0  DOWNLOADED

=====================================================
Config ROM for slot: 3

 0  0  1  0  0  0 8c 7a  4  8 49 30 bc df 9c e3
be df  5  0  0  0 a1 1e  6  8 80  9  9  8 f0 79
 4  8 68 7a  4  8 52 34 bd df  b f2 a7 df e2  3
 0  0 e8 d1 be df f7  b a8 df 3b  6  0  0  b f2
a7 df 1c 7a  4  8 ed 56 bc df  b f2 a7 df  b f2
a7 df d0  4 b3 df d0  4 b3 df e8 d1 be df 3f  8
 0  0 20 3e  0  0 5f f2 a7 df 30 c0 be df d0  4
b3 df bc c0 be df  0  0  0  0 e8 d1 be df  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0 dd 79 bc df e8 d1
be df 5c 7a  4  8 89 7c bc df  1  0  0  0 e8 d1
be df 74 7a  4  8 1b b4 bc df  1  0  0  0 bc c0
be df d0  4 b3 df e8 d1 be df c4 7a  4  8 f1 f5
bc df bc 7b  4  8 d4 7a  4  8  0 30 b0 df d0  4
b3 df  a f2 a7 df  0  0  0  0 d0  4 b3 df 9c e3
=====================================================

Checksum        : 0x0
Vendor ID (high): 0x4
Vendor ID (med.): 0xBD
Vendor ID (low ): 0xA7
Vendor is       : UNKNOWN

Primary Board ID-3: 0x0  (Unknown)
Primary Board ID-2: 0xBE  (Unknown)
Primary Board ID-1: 0xA8
Primary Board ID-0: 0x0
MODEL: (0xA800) UNKNOWN

Secondary Board ID-3: 0xA7
Secondary Board ID-2: 0x4
Secondary Board ID-1: 0xBC
Secondary Board ID-0: 0xA7
```

Figure 23. Listboards Output on Linux

To use listboards on Windows, you must start the boards using the Dialogic Configuration Manager (DCM). Then invoke *listboards* from the Command Prompt window. Figure 24 shows the results of running *listboards* on a Windows system.

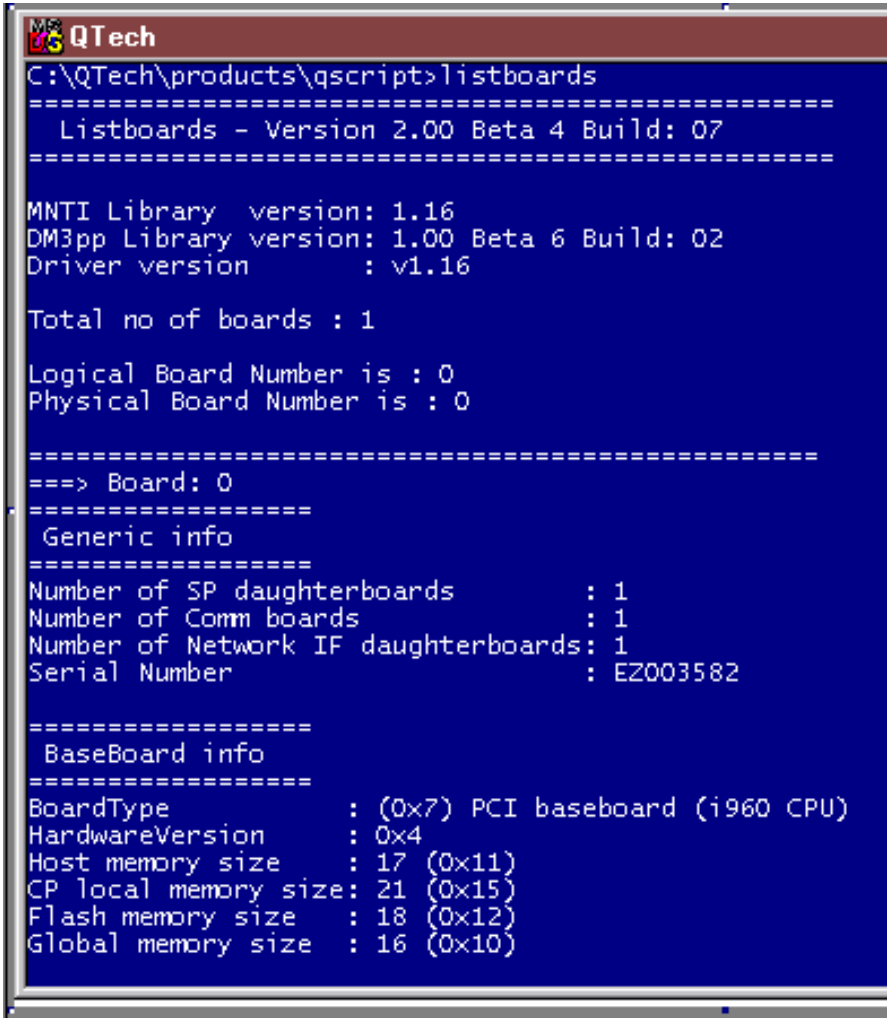

Figure 24. Listboards Output on Windows

3.17. MercMon

MercMon provides counter information about DM3 device drivers (Class Driver and Protocol Driver). These drivers maintain various counters that can aid in monitoring system activities and interpreting system behaviors.

Operating Systems: Windows

Command Line: `mercmon [parameter_list]`

The *MercMon* utility uses the following command line parameters:

Parameter	Description
/f <*file*>	Log file name (default is mercmon.log)
/t<*n*>	Display Timer. The time interval (in milliseconds) between each screen refresh (default is 1000).
/l<*n*>	Logging Interval. The time interval (in seconds) between each log file update (default is 600).
/w	Enable/disable logging to a file (default is enabled)
/?	Displays the program help screen (optional)

Example: This example enables logging to the file *mercmon.log* every five minutes and refreshes the screen every two seconds:

```
mercmon /t 2000 /l 300
```

Driver Counters: For counter details, see Table 5 and Table 6.

NOTE: All counters are on a per board basis except for *failMpathFind* and *failStrmFind*. These two counters are on a global basis.

Table 5. Class Driver Counters

Class Driver Counter	Description
CompleteReads	Returns the size (in Bytes) of all the Read requests completed by the Protocol Driver (i.e. the actual size in KB of all the data read from the board)
CompleteSends	Returns the size (in Bytes) of all the Messages sent to or received from the board
CompleteWrites	Returns the size (in Bytes) of all the Write requests completed by the Protocol Driver (i.e. the actual size in KB of all the data written to the board)
FailMpathFind	Returns the number of times a request to get a Message Handle (Mpath) failed (global counter)
FailStrmFind	Returns the number of times a request to get a Stream Handle failed (global counter)
NumCanTakes	Returns the number of “Can Takes” received for that board
NumReads	Returns the total number of Read requests sent down to the Protocol Driver
NumSends	Returns the total number of Messages sent down to the Protocol Driver
NumWrites	Returns the total number of Write requests sent down to the Protocol Driver.
NumWriteSplit	Returns the number of write requests, which were split for that board
NumOpenedStreams	Returns the number of Open Streams on that board at any given time
NumCloseStreamErr	Returns the number of Stream close requests failed on that particular board

Table 5. Class Driver Counters (Continued)

Class Driver Counter	Description
NumOpenStreamErr	Returns the number of Stream open requests failed on that particular board
NumStreamClose	Returns the number of Stream close requests on that particular board
NumStreamOpen	Returns the number of Stream open requests on that particular board
SizeReads	Returns the size (in Bytes) of all the Read requests posted down to the Protocol Driver
SizeSends	Returns the size (in Bytes) of all the messages sent down to the Protocol Driver
SizeWrites	Returns the size (in Bytes) of all the Write requests sent down to the Protocol Driver
TimeoutReads	Returns the number of Read requests that timed out
TimeoutSends	Returns the number of Messages that timed out while waiting to be sent to the board or awaiting a reply from the board
TimeoutWrites	Returns the number of Write requests that timed out

Table 6. Protocol Driver Counters

Protocol Driver Counter	Description
MsgsInPerSramSession	Returns the number of messages read from the SRAM in one session
MsgsOutPerSramSession	Returns the number of messages written to the SRAM in one session
NoInDataDPCisr	Returns the number of times when we received an interrupt but there was no data transfer between the SRAM and the HOST

Table 6. Protocol Driver Counters (Continued)

Protocol Driver Counter	Description
StrmsOutPerSramSession	Returns the number of streams written to the SRAM in one session (i.e., number of data blocks written)
StrmsInPerSramSession	Returns the number of streams read from the SRAM in one session (i.e., number of data blocks read)
TotalAsyncMsgQDone	Returns the number of requests completed in the AsyncMsgQ
TotalBadSramAddr	Returns the number of times we tried reading a message/data (Streams) from the SRAM but the SRAM address was wrong
TotalBadSramAddrAlloc	Returns the number of times the system tried to allocate a data block but the address was wrong
TotalBadSramAddrRelease	Returns the number of times the system tried to release a data block but the address was wrong
TotalBadSramDataCount	Returns the total number of times we got a data block of size greater then the SRAM_DATA_BLOCK_SIZE
TotalBadSramOffset	Returns the number of times we tried reading data (Streams) from the SRAM but the offset calculation was incorrect
TotalBadSramOffsetAlloc	Returns the number of times when we allocated a data block but offset was incorrect
TotalBigMessagesRcvd	Returns the number of times we read a “BIG” message from the SRAM (i.e. message size greater then 24 bytes)
TotalBigMessagesSent	Returns the number of times we wrote a “BIG” message to the SRAM (i.e. message size greater then 24 bytes)
TotalBogusInterrupts	Not used

Table 6. Protocol Driver Counters (Continued)

Protocol Driver Counter	Description
TotalDpcOverruns	Returns the total number of DPC overruns (i.e. we were not expecting an Interrupt but we got one)
TotalDmaInterrupts	Returns the total number of interrupts received from the board for performing DMA
TotalFatSramBlocks	Returns the number of times the system received a “CHAINED” data block (i.e., the data blocks are linked together)
TotalIrpsCancelled	Returns the total number of canceled requests in any queue
TotalMsgInQ	Returns the number of requests in the Message In Queue, which are awaiting a reply from the board
TotalMsgInQDone	Returns the number of requests completed from the Message In Queue (i.e., requests completed and sent to the application)
TotalMsgInSram	Returns the total number of messages read from the SRAM
TotalMsgOutQDone	Returns the number of requests completed from the Message Out Queue (these requests are sent to the board and then moved to the Message In Queue, if expecting a reply, or completed and sent to the application)
TotalMsgOutSram	Returns the total number of messages written to the SRAM
TotalMsgOverruns	Returns the number of Overruns for the Orphan Message Queue (i.e. we had an Orphan Message but there was no space in the Orphan Message Queue)

Table 6. Protocol Driver Counters (Continued)

Protocol Driver Counter	Description
TotalMsgTimeouts	Returns the total number of requests timed out while waiting to be written to the SRAM or awaiting a reply from the SRAM (i.e. either in the Message In Q, Message Out Q or the Async Message Q)
TotalOrphanMsgsv	Returns the total number of Orphan Messages (i.e., Messages which were read from the SRAM but do not have any pending requests from the application)
TotalOrphanMsgMatches	Returns the total number of messages matched in the Orphan Message Queue
TotalOrphanMsgVolume	Returns the size (in bytes) of the messages present in the Orphan message queue at any given time
TotalOrphanStrms	Returns the total number of Orphan Streams in the Orphan Stream Table (i.e., data which was read from the SRAM but did not have any pending Read requests from the application)
TotalOrphanStrmMatches	Returns the total number of Streams matched in the Orphan Stream table
TotalOrphanStrmVolume	Returns the size (in bytes) of the data present in the Orphan Stream table at any given time
TotalSramDataFull	Returns a count of the number of times the HOST tried to write a stream (data) to the SRAM but the SRAM was full
TotalSramGrantInterrupts	Returns the number of “HOST SRAM PENDING” interrupts
TotalSramGrantLost	Not used

Table 6. Protocol Driver Counters (Continued)

Protocol Driver Counter	Description
TotalSramInterrupts	Returns the total number of “NORMAL” interrupts received from the board (i.e. there is something to read from the SRAM)
TotalSramMsgFull	Returns a count of the number of times the HOST tried to write a message to the SRAM but the SRAM was full
TotalStrmInQ	Returns the number of Read requests pending in the Stream In Queue
TotalStrmInQDone	Returns the number of Read requests completed from the Stream In Queue (i.e., Read Requests completed and sent to the application)
TotalStrmInSram	Returns the number of data packets read from the SRAM
TotalStrmOutQ	Returns the number of Writes performed by the user on that particular board. It is also incremented whenever the application sends an End Of Stream command.
TotalStrmOutQDone	Returns the number of Write requests completed from the Stream Out Queue (i.e., Write Requests completed and sent to the application)
TotalStrmOutSram	Returns the number of data packets written to the SRAM
TotalStrmOverruns	Returns the number of Overruns for the Orphan Stream Table (i.e. we had an Orphan Stream but we had exceeded the maximum amount of memory allocated for the Orphan Stream Table or we could not allocate memory.
TotalStrmTimeouts	Returns the total number of Read or Write Requests timed out (from the Stream In Queue or the Stream Out Queue)

Table 6. Protocol Driver Counters (Continued)

Protocol Driver Counter	Description
TotalUnknownInterrupts	Returns the total number of unknown interrupts received from the board

3.18. PDK Configuration Utility

The *PDK Configuration Utility* is used when integrating E-1 R2MF protocol information (CDP files) into IPLink, Fax or HDSI FCD files. The utility is used to **integrate** the CONFIG and CDP files into an updated FCD file. The utility can be used to edit the files as well as to generate the FCD file.

For a detailed procedure, refer to *Modifying the FCD File Parameters Using the PDK Configuration Utility* in the *DM3 Configuration File Reference*. For detailed information on the parameters contained in the CDP file, see the *GlobalCall Country Dependent Parameters (CDP) Reference*.

Operating Systems: Windows, Linux

3.19. PDKManager Utility

The *PDKManager Utility* is a command line utility used to download and configure E-1 R2MF protocol information on QuadSpan or DualSpan boards. The utility is invoked after all other CONFIG file changes have been made and downloaded to the board.

For a detailed procedure, refer to *Downloading CDP File Parameters Using the PDKManager* in the *DM3 Configuration File Reference*. For detailed information on the parameters contained in the CDP file, see the *GlobalCall Country Dependent Parameters (CDP) Reference*.

Operating Systems: Windows, Linux

3.20. Phone

The *Phone* utility uses TSC and ToneGen instances and requires a TSC component. The *Phone* utility can control a single DM3 resource channel (make calls, wait for calls, etc.), monitor channel and call states, and send call control operations to the DM3 GlobalCall resource.

Operating Systems: Windows, Linux

Command Line: `phone [parameter_list]`

The *Phone* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional)
-chan <*n*>	Channel number (optional)

Example: This example runs the *Phone* utility on board 0, line 1, channel 1:

```
phone -board 0 -line 1 -chan 1
```

Display: Figure 25 shows the *Phone* display.

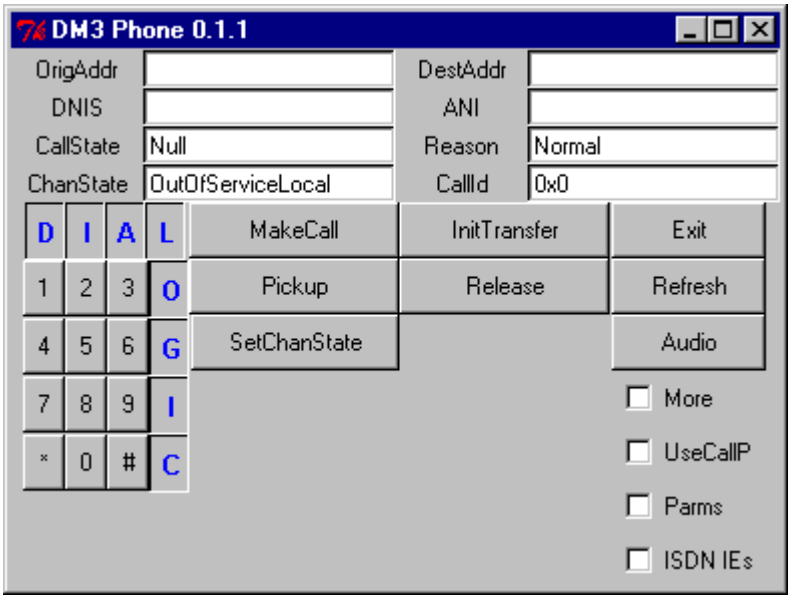

Figure 25. Phone Display

3.21. QError

QError displays a string associated with the error code returned in a *QResultError* message from the board. *QError* displays error code strings for DM3 Kernel errors only.

The error code string is generated with a Perl script directly from the header file files, so it is as accurate as the comments in the header files. The source to the *QError* code is arranged so that the "get error string" function can be pulled out and used in any application.

Operating Systems: Windows, Linux

Command Line: `qerror [parameter_list] ERRORCODE`

The *QError* utility uses the following command line parameters:

Parameter	Description
-b<*n*>	Base of input number (hexadecimal is 16, decimal is 10, default is 16)
-d<*level*>	Do not modify. Leave at default value of 0.
-h	Displays the program help screen (optional)
-v	Displays the program version (optional)

Example: `qerror 28008`

Output: The output from this example would be as follows:

```
ERROR==> 0x28008 (163848)

Kernel Error:

  Cluster does not exist or cannot be found
```

3.22. STD Config

STD Config provides a flexible way to configure DM3 component parameters. You put the parameters to be set and retrieved for a particular component into a file. You can create and modify these files. When used in conjunction with the *STD Config* utility these component parameters (for example, lineadmin, CCS, player) can be easily configured.

Operating Systems: Windows, Linux

Command Line: `stdconfig [parameter_list]`

The *STD Config* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.

Parameter	Description
-file <*name.ext*>	The file name of the containing the relevant parameters of the component to be configured (such as tsc.prm, ccscomp.prm).
-inst <*n*>	Specifies the particular instance (of the component) whose parameters the user wants to modify.
-comptype <*n*>	Standard Dialogic component types (1 - 255).
-class <*name*>	One of the standard Dialogic components (such as TSC, LCON, CHP). This should match the relevant -file parameter.

Example: This example runs the *STD Config* utility on board 0:

```
stdconfig -board 0
```

Display: Figure 26 shows the *STD Config* display.

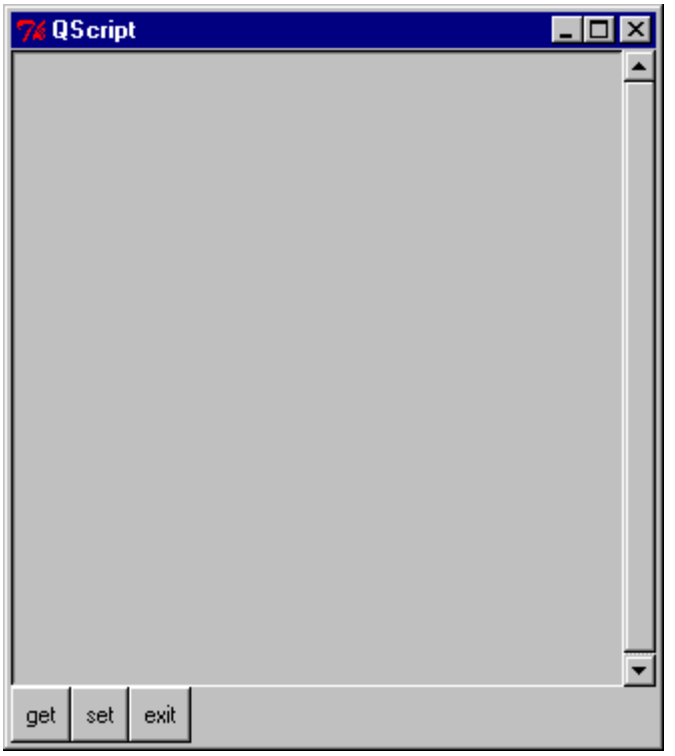

Figure 26. STD Config Display

3.23. StrmStat

StrmStat displays the current state of the specified stream(s).

States shown include:

- closing
- closed
- close failed
- opening

- opened for write
- opened for read
- open failed.

Operating Systems: Windows

Command Line: `strmstat [parameter_list]`

The *StrmStat* utility uses the following command line parameters:

Parameter	Description
/b <*n*>	Board number (required, default is 0). Use the *Listboards* utility to obtain the board number.
/f <*file*>	Log file name (default is *strmstat.log*)
/i <*n*>	Number of streams on which to report (default is 120)
/s <*n*>	The first stream to report (default is 1)
/?	Displays the program help screen (optional)

Example: This command displays the stream state for stream IDs 1 to 16 on board 0:

```
strmstat /b 0 /i 16
```

Output: The output is similar to the following example:

```
   Stream Status for Board = 0
Stream ID       State           Additional Info
     1          Closed
     2          Closed
     3          Closed
     4          Closed
     5          Closed
     6          Closed
     7          Closed
     8          Opened          Write
     9          Opened          Write
    10          Opened          Write
    11          Opened          Write
    12          Opened          Write
    13          Opened          Write
```

```
    14          Opened          Write
    15          Opened          Write
    16          Opened          Write
TOTALS
Closed = 7      Opened = 9      Closing = 0     Opening = 0
CloseErr = 0    OpenErr = 0     Unknown = 0
```

3.24. TSP Config

TSP Config sets and retrieves DM3 protocol variant parameters. Using this utility the user can change protocol variant parameters dynamically from one call to the next.

This tool can be used on any DM3 product that has a T1 CAS or T1/E1 ISDN TSP resource. The tool does not work for R2MF protocols or Analog protocols.

Operating Systems: Windows, Linux

Command Line: `tspconfig [parameter_list]`

The *TSP Config* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-id <*n*>	Protocol variant id (1 - 32), whose parameters you are configuring dynamically. For example, tspconfig -board 0 -id 2, will display the parameters of protocol variant id 2.

Example: This example runs the *TSP Config* utility on board 0:

```
tspconfig -board 1
```

Display: Figure 27 shows the *TSP Config* display.

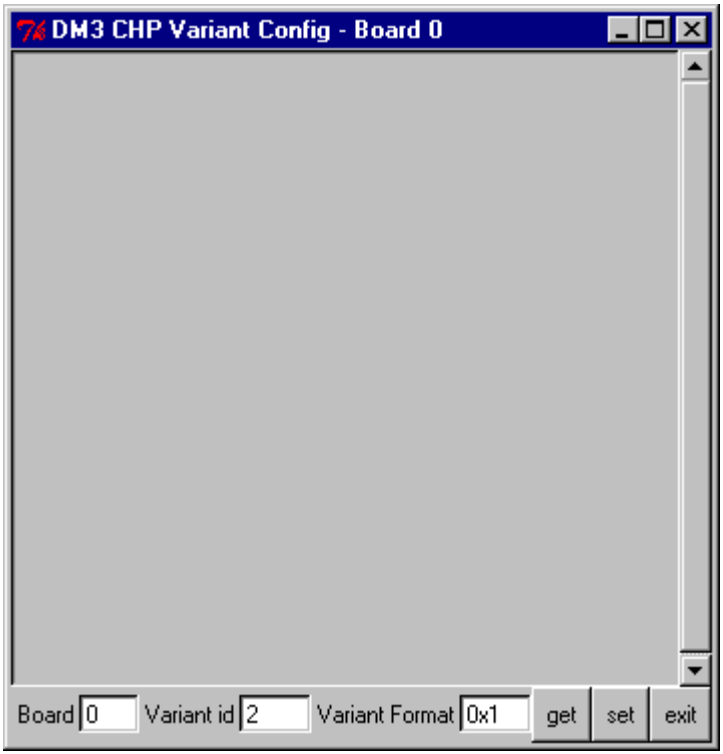

Figure 27. TSP Config Display

3.25. TSP Monitor

TSP Monitor performs the following:

- Monitors one or two DM3 GlobalCall resource channels.
- Traces all levels of the protocol including:
  - DM3 GlobalCall resource call control operations from clients
  - DM3 GlobalCall resource call state changes
  - DM3 GlobalCall resource channel state changes

  - CAS signaling bits
- Launches an audio tool for recording/playback purposes on the channel(s). It plays audio data (a file from the host) using the TSP.
- Checks timing via point and click on GUI (Figure 28)
- Tunes protocols and identifies configuration problems

Operating Systems: Windows, Linux

Command Line: `tspmon [parameter_list]`

The *TSP Monitor* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-chan <*n*>	Channel number (optional, default is 1)

Example: This example runs the TSP Monitor utility on board 0, line 1, channel 1:

```
tspmon -board 0 -line 1 -chan 1
```

Display: Figure 28 shows the *TSP Monitor* display.

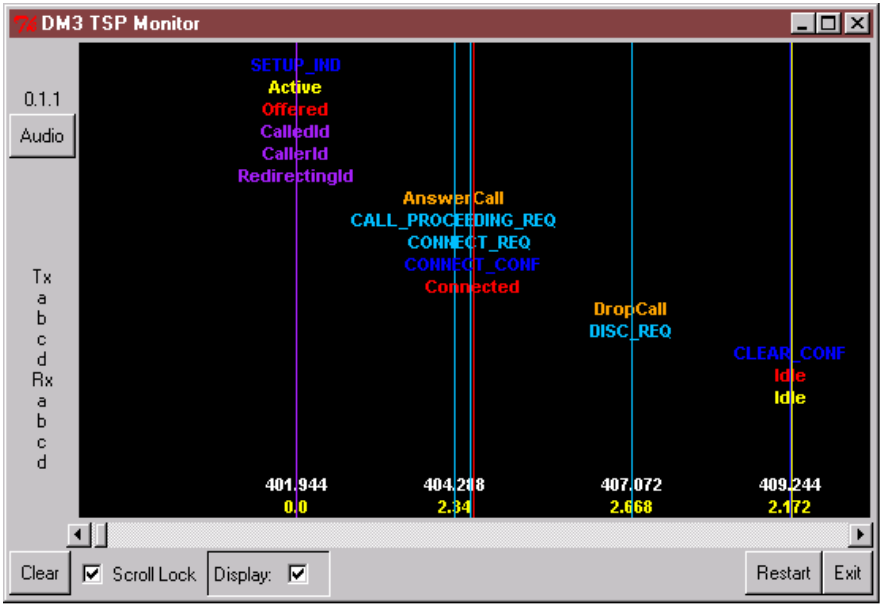

Figure 28. TSP Monitor Display

3.26. TSP Tracer

TSP Tracer traces CAS protocols with timing information and saves trace information to a log file. The *TSP Tracer* utility does not support ISDN protocol tracing. For ISDN protocol tracing, use the *TSP Monitor* utility, which supports all protocols.

Operating Systems: Windows, Linux

Command Line: `tsptrace [parameter_list]`

The *TSP Tracer* utility uses the following command line parameters:

Parameter	Description
-board <*n*>	Board number (required). Use the *Listboards* utility to obtain the board number.
-line <*n*>	Line number (optional, default is 1)
-chan <*n*>	Channel number (optional, default is 1)

Example: This example runs the TSP Tracer utility on board 0, line 1, channel 1:

```
tsptrace -board 0 -line 1 -chan 1
```

Display: Figure 29 shows the *TSP Tracer* display.

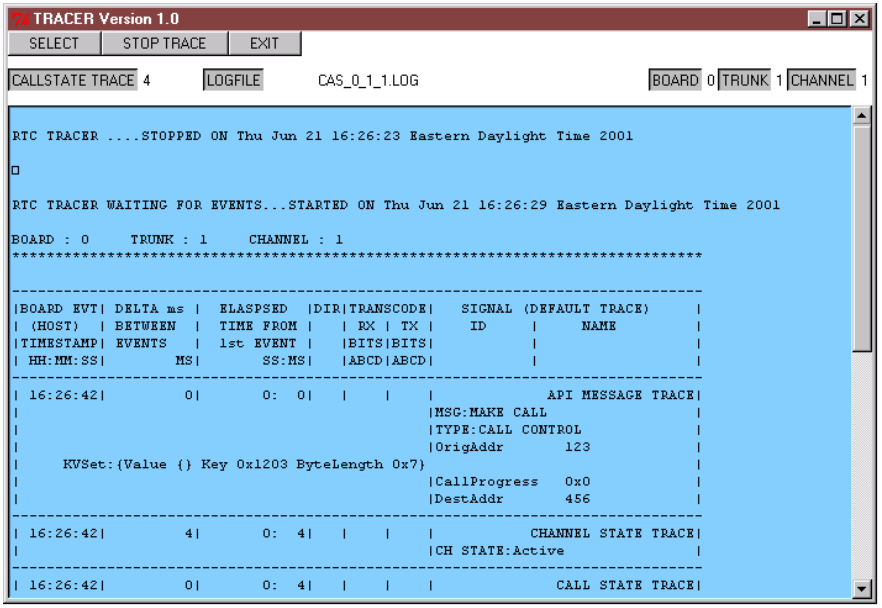

Figure 29. TSP Tracer Display

4. Troubleshooting

This chapter describes how to get help from Dialogic if the hardware or software you have is performing incorrectly.

4.1. Information Needed by Technical Support

The following information should be collected prior to contacting Dialogic Technical Support:

- basic host information such as operating system and version, CPU, and memory size
- which Dialogic System Release is installed (including any additional Service or Feature Packs)
- which Dialogic products are installed
- basic DM3 information such as version numbers for CP, SP, drivers, PCD/FCD files, number of boards installed, any event viewer errors (e.g., assert string), and number of downloads performed
- any log or capture file created by the diagnostic utilities
- POST failure code, if applicable

4.2. Contact Information

Contact Dialogic Technical Support at *http://support.dialogic.com*.

4.3. RMA Information

For RMA details, refer to the hardware *Quick Install Card* shipped with each Dialogic board.

Index

O

P

Q

R

S

T