XDS Windows NT H.110 Driver Reference Manual

Version 2.2
December 2003

American Tel-A-Systems, Inc.
258M003B ©

This page was intentionally left blank.

Contents

5A XDS MVIP-95 Software Interface Description

5B XDS MVIP-95 Command Reference

6A XDS CT-BUS Software Interface Description

6B XDS CT-BUS Command Reference

7 MVIP-Related Error Codes

XDS Windows NT H.110 Driver Reference Manual

American Tel-A-System, Inc.
http://www.amtelco.com/
608-838-4194
4800 Curtin Drive McFarland, WI 53558, USA
258M003B

Driver Package Software Installation and Removal

This page was intentionally left blank.

1.0 Driver Package Installation

One prerequisite for the XDS driver to be installed is that Microsoft Windows NT Service Pack 3 or higher be installed on the system.

If the XDS hardware has not been installed in the system yet, please refer to the appropriate hardware reference manual and install the hardware first. Once the hardware has been installed properly, the user may proceed with the software / driver installation.

Each cPCI board uses 8K of memory. The resources for cPCI boards can be viewed in the system BIOS at boot-up. Be sure that there will be a hardware interrupt available for a PCI device. When Windows NT starts up, it will assign the memory offset, IRQ, and I/O port dynamically. These settings may be viewed by using the "Windows NT Diagnostics" tool in the *Administrative Tools* menu.

The XDS H.110 Windows NT Driver Package comes in the form of one CDROM disc (Amtelco part number 258CD000). The driver package contains the driver, an installation utility, test and demonstration programs, and the source code for all of the included programs. To install the driver, insert the CDROM. If the driver package installation does not start automatically, it will need to run manually. This can be done by running setup.exe (on the CDROM) from '**RUN'** on the start menu or Add/Remove Programs from the Control Panel. The WISE installation program will guide the user through the rest of the procedure. This will create the all the necessary directories, copy files to the hard drive, make the necessary modifications to the Windows NT registry, and add the *Amtelco H.110 Technology* folder to the Start Menu Programs. When setup has successfully completed, a new window will appear on the desktop containing five XDS program icons. These will be labeled "xdsinst_h110", "xdsutil", "signal_test", "sig_util", and "xds_bri_config". Note that setup does NOT install the low-level device driver for the boards. That procedure is described in the next section (Driver Installation), and must be done by the user manually.

2.0 Low-level Driver Installation

The XDS H.110 device driver is installed by double clicking on the “xdsinst_h110” icon in the *Amtelco H.110 Technology* window or by executing the “xdsinst_h110.exe” application from the command line. The default path for this program is:

<drive>:\Program Files\Amtelco\XDS_H110\Xds\Bin\Intel

As with all device drivers in Windows NT, the user must have NT administrator privileges in order to install the low-level driver.

Select the driver(s) to be installed. The **XDS** driver is required for communication to any XDS board(s) in the system. The **HS** hot swap driver will need to be installed if your application will use any form of “hot-swapping” of any of the XDS boards in the system.

If both drivers are to be installed, the XDS driver should be installed first, and then the hot-swap driver second.

Now, select the driver to be installed and click on the “Go” button. A pop-up screen will appear to report the outcome of the device driver installation. The ID strings and offset (device number) for all XDS boards in the system should appear in the middle test box. If board(s) are physically present, but not listed in the display box, there may be a problem with the settings or the board(s).

There is a text box at the bottom of the window for each driver. When each driver is loaded and started, the respective text box will display “LOADED”. These boxes will be empty (blank) when the respective driver is not started.

If problems occur during the driver installation, they will be identified in the Windows System Log. These can be retrieved by:

1) From the Windows **Start**, open “Administrative Tools”.
2) Open the “Event Viewer”.
3) Ensure the title bar indicates “System Log”. If not, click on Log, then click on System.
4) Events should appear, by default, in order from newest to oldest.

Error Event Codes include:
Event Code of 1 indicates that messages can not be sent to a board
Event Code of 2 indicates messages are not being received from a board
Event Code of 3 indicates an interrupt failure

3.0 Low-level Driver Removal

The program **xdsinst_h110** is also used to unload or uninstall the XDS and/or Hot-swap device driver. The user may unload (stop) a selected driver or uninstall (remove) a selected driver with this utility.

Unload –
To stop a selected driver from running, check the driver box and the Unload box and click on the "Go" button. This will stop the selected driver and set the startup mode for it to manual. In order to use/start the driver again, the user may use **xdsinst_h110** as described in section 2.0 (as if it had not yet been installed) or one of the methods described in section 4.0.

Uninstall –
To stop and uninstall a selected driver, check the driver box and the Uninstall box and click on the "Go" button. This will stop the selected driver from running, remove the driver from the Windows NT Devices, and remove the driver itself from the winnt\system32\drivers directory. In order to use/start the driver again, the user must use **xdsinst_h110** as described in section 2.0 to install it.

If both drivers are to be un-installed, the hot-swap driver should be un-installed first, and then the XDS driver second.

4.0 Low-level Driver Control

The XDS and hotswap device driver(s) are visible in the Windows NT Device Manager - “Devices”. “Devices” is located in the **Control Panel**. The driver(s) should be automatically started every time the machine reboots. Each driver can also be started and stopped using the following commands at a command prompt:

net start xds_h110 - starts the XDS driver
net stop xds_h110 - stops the XDS driver

net start hsdriver - starts the hot swap driver
net stop hsdriver - stops the hot swap driver

They may also be controlled in “Devices” by highlighting each one and selecting the appropriate control.

5.0 Driver Package Removal

If section 3.0 (driver un-installation) has not been completed, do that now. When the user wishes to remove the XDS software from their system, they may do so by running Add/Remove Programs. This again, is located in the Windows NT Control Panel. Select the “Amtelco XDS Windows NT H.110 Driver Package” software package by highlighting it in the list box. Then, click on the **Add/Remove…** button. It will then continue and inform the user of any choices available.

If files have been modified in any way, added, or removed from any of the driver package directory folders, the user will be notified that some of the contents were unable to be removed. To ensure clean removal, delete any remaining files/folders in the \Program Files\Amtelco directory.

This page was intentionally left blank.

Driver Package
Programs and Source Code

This page was intentionally left blank.

1.0 XDS Source Code Description

All of the source code used to build the programs, DLLs, and driver has been included for the user's convenience. If any or all of the code is "re-used", the American Tel-A-Systems, Inc. copyright information must be included with it. All of the project workspaces for this release package have a pre-processor define ("XDS_H110") in them, due to the fact that many of the projects included may work with other XDS driver packages. Microsoft Visual C++ 6.0 (32 bit) was used to create, compile, and build all of the applications included. Microsoft Visual C++ 6.0 (32 bit) and the Microsoft Windows DDK was used to compile and build the low-level driver (xds_h110.sys).

2.0 Tests / Utilities

All message strings sent to any board, using any one of the provided utilities, must be in CAPITOL letters.

XdsInst_H110 (GUI Driver Installation Utility)
A Graphical User Interface, XdsInst_H110, has been provided to ease the installation procedure of the XDS low-level driver. It is also a quick way to view what boards and the board number for each of them are operating properly. They will be displayed automatically in the list box. Detailed instructions on how to use this utility to install a driver is described in section 2.0, "Low-level Driver Installation", in the driver installation part of this manual.

XdsUtil (GUI Utility)

A Graphical User Interface, XdsUtil, has been provided for simple and user-friendly communication with XDS boards. There is a pull-down list used to select which board to transmit messages to, and one to display the received messages from. Message strings sent are typed in the edit box above the message receive list box. Boards that have physical interface ports will display the port states in the port state window on the right. BRI boards will display the Layer 1 port states in this window. The port-range displayed may be controlled by changing the port range spin control. Click ion the right arrow to show the next block of ports, and to go back click on the left arrow. This program uses message polling to receive messages from the driver. A button labeled "Show Boards" is used to display a list of present boards. A modal dialog box will appear, when finished, click on the "Done" button and you will return to the main dialog. The "Clear Message Window" button simply clears the messages displayed in the message receive list-box.

Like all of the XDS applications, it should be used alone and not in combination with any other XDS programs or utilities. Opening an application while one is already running may result in message passing problems.

Driver Command Line Test

The test program **test_drv** is a simple program, written in 'C', that demonstrates how to make driver calls. It is a text-based command line application that makes IOCTL calls directly to the driver. The syntax is "**test_drv n"**, where n is the number of an installed XDS board. The program first displays any messages that might be already on the board's queue(s). To send a message, type in '**s**' and then the command string followed by pressing the "Enter" key. To receive any messages that might be on the message or query queue, type in '**r**' and then the "Enter" key. The responses along with any messages on the board will be displayed on the screen for the user. The user may choose to switch boards at any time during run-time, by using the 'b' option to "select a different board to communicate with" .To quit the program, type in '**q**' and then press the "Enter" key. Any messages on the queues will be displayed and then the program will terminate.

Signaling Mechanism Test Utilities

Several programs are available to test and illustrate how the signaling capability of the driver. It is a more efficient method of message handling from the driver. Once the driver receives a message from the board, it arms the signaling mechanism notifying the application of a message to be received. **Sig_Util** is a GUI based program that allows the user to communicate with any XDS board. It is similar to XdsUtil, in the respect that it can also be used to send and receive messages from boards. The user has the option of disabling the signaling mechanism in the application. This is done by turning off the "Notify the main thread when interrupts happen" option (checkbox). The user also has the option of not retrieving the message(s) when interrupted. This can be done by turning off the "Retrieve message when interrupted" option (checkbox). Sig_Util has one drop-down list to select the board to be used. Once that the board is selected, you may communicate with it by typing in message strings in the in the edit box above the message receive list box. To send it, press the "Enter" key or click on the "Send" button. The "Receive" button is included to manually receive messages when the "Retrieve message(s) when interrupted" option is turned off. Another button is present, labeled "Test All", that will send an array of test messages to the board. The results and receive messages, again, will be displayed in the receive message list-box. A button, label "Layer 3 Msg", is used to demonstrate the transmission of a Layer 3 message to BRI boards. It is intended for use with BRI boards only. To exit the program, click on the "Exit" button.

The program **signal_test** sends a command repeatedly to a selected board (from drop-down list) when the "Start" button is pressed. Nothing will be displayed on the screen while messages are being sent. When the "Stop" button is selected, the program will stop sending messages and will count the received responses. It will then verify if the number of responses differs from the number of messages sent is the same. The program will display the results and statistics for the user. When finished, click on the "Exit" button to exit and close the program.

DLL Command Line Tests

XdsLib110

The test program **test_dll.exe** is an example of how the XDS H.110 Native DLL can be linked to a program and tested. All of the functions in this program are included in the XdsLib110 DLL. The syntax is "**test_dll n"**, where **n** is the number of an installed XDS board, or if the user is not sure of which board to communicate with, they may simply type in "**test_dll"** at the command line, and the program will list the board IDs and numbers of boards available. The program will first display any messages that might be already on the board's queue(s).

Then, the options: **s = send a command / message to the board**, **b = select a different board to communicate with**, **l = send Layer 3 test message (BRI boards only)**, **r = receive messages from the board**, or **q = quit**, are displayed. To send a message, type in '**s**' and then the command string followed by pressing the "Enter" key.

To receive any messages that might be on the message or query queue, type in '**r**' and then the "Enter" key. The responses along with any messages on the board will be displayed on the screen for the user. Any messages on the queues will be displayed and then the program terminates.

XdsMv90

The test program **testmv90.exe** is an example of how to open XdsMv90.dll and make SwDevIoctl calls to it. The test program syntax is "**testmv90 n"**, where n is the number of an installed XDS board. The **testmv90** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

XdsMv95

The test program **testmv95.exe** is an example of how to open XdsMv95.dll and make SwDevIoctl calls to it. The test program syntax is "**testmv95 n"**, where n is the number of an installed XDS board. The **testmv95** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

XdsCtBus

The test program **testctbus.exe** is an example of how to open XdsCtBus.dll and make SwDevIoctl calls to it. The test program syntax is "**testctbus n"**, where n is the number of an installed XDS board. The **testctbus** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

PCI Bus / Device Number Demo

A program called **XdsPciRes** was written to display the PCI bus and device (function) number for each XDS board in the system, along with each board's geographic number (if supported – if not this number will be arbitrary but different than every other board in the system) and board ID code. It is run by simply typing in **XdsPciRes** (with no arguments) at a command line.

Hot-plug (Admin) User Demonstration

The **tstchs** program demonstrates to the user how and what upper-level (DLL) XDS function calls to use to remove a running XDS H.110 board and inserting a replacement XDS H.110 board in place of a removed XDS H.110 board in a running system. The replacement board MUST be inserted in the same physical slot as the one removed and the program, **tstschs**, must remain running while the board is being replaced.

The syntax is "**tstchs n"**, where **n** is the number of an installed XDS board, or if the user is not sure of which board to communicate with, they may simply type in "**tstchs"** at the command line, and the program will list the board IDs and numbers of boards available. The program will first display any messages that might be already on the board's queue(s).

Then, the options: **s = send a command / message to the board**, **b = select a different board to communicate with**, **d = remove board**, **i = insert board (into same slot as a board has been removed from), r = receive messages from the board**, or **q = quit**, are displayed.

Tstchs can send and receive messages just as test_dll, but has facilities for removing a board and replacing it with a new one (of the same type, in the same physical slot). Before removing an XDS board, the user should be aware that any messages on the board's message queues will be lost when the board is removed and the replacement board is inserted. Also, if any special configuration information was saved on the board (BRI board), the user may want to save it to disk (if possible), by using XDS_BRI_Config.

To remove an XDS board, the user will run the **tstchs** utility from a command line (as described above). The user will then use the '**d**' option to "remove board". When the board is ready for removal, the blue hot-swap LED on the front panel of the front board will turn on. The user may now remove the front board **first** (consult the hardware reference manual for detailed hardware removal instructions), and then the rear board (if there is a rear I/O board). Under **NO** circumstances should the user remove the rear I/O board will the front board is in a running system.

To replace the XDS board, the user will insert the replacement rear I/O board first (if there is one), and then the front board (consult the hardware reference manual for detailed hardware insertion instructions) in the same slot as the board that was removed. Once the replacement board is in the same physical slot as the board removed, the user will use the '**i**' option to "insert board". After the replacement board is the system, the user should send and receive messages, by using the '**s**' and '**r**' options, to test that the process and replacement board work. Once the board is back up and running, the user may then exit the program, by using the '**q**' option.

3.0 Downloader

All of the XDS H.110 boards are equipped with flash memory, which contains the board program. New revisions of the program can be downloaded to this memory using the downloader program **wn386dlc**. To use this program, the driver must be started and recognize the board. The program to be downloaded is contained in a .hex file. This file will include a header identifying the board type so that it can only be loaded onto a compatible board. The syntax for the downloader is:

wn386dlc <hexfile.hex> <board number (decimal)>

where the segment specified is either a 'C' for the control processor or 'D' for the DSP processor. For example

wn386dlc 258H001.HEX c 16

will flash the firmware file, 258H001.hex, to the control processor onto board number 16.

4.0 DLL Descriptions

XdsLib110 DLL
An "XDS" DLL (xdslib110.dll) has been provided to access XDS native board functions. These include proprietary functions for use with XDS boards. Many of the applications in this package use this DLL. When creating a new application, be sure to link in XdsLib110.lib in the project workspace. Details of the functions included in this library may be found in the document *XDS H.110 Library Reference Manual, 258M013.*

XdsMv90
The DLL provides high-level native XDS and MVIP-compliant commands along with the mandatory scope of MVIP-90 commands. A listing and description of each of these commands is included with this reference manual, in the "MVIP-90 Software Interface Description" section.

XdsMv95
The DLL provides high-level native XDS and MVIP-compliant commands along with the mandatory scope of MVIP-95 commands. A listing and description of each of these commands is included with this reference manual, in the "MVIP-95 Software Interface Description" section.

XdsCtBus
The DLL provides high-level native XDS and MVIP-compliant commands along with the mandatory scope of CT-BUS commands. A listing and description of each of these commands is included with this reference manual, in the "CT-BUS Software Interface Description" section.

5.0 Hotswap Test / Utilities

The user must remember that "hotswap" and "hotplug"/"hot-admin" can **NOT** be used together.

All of the hotswap driver test and service utilities are located in the \hotswap\bin\intel directory in the driver package. Again, as described in the Driver Installation section, the hotswap driver **hsdriver.sys** can be installed using the X**dsInst_H110** installation utility (found in the \XDS\bin\intel directory).

Hotswap Driver Utility

The program **hsdtest** is used to test various facilities of the hotswap driver. It can be found in the \hotswap\bin\intel directory. This program may be run at a command prompt by simply typing in "hsdtest". Choices '0' – '8' are the hotswap test modules that test the hotswap portion of this driver. Choices 'A' – 'C' are the XDS test modules that test the XDS communication portion of this driver.

Hotswap Manager

The hotswap manager is used by all of the H.110 boards, in a system that has hot swap capabilities. This is a WIN32 service, which automatically runs in the background and manages all of the hot swap activity in the system. The executable for the service can be found in the \hotswap\bin\intel directory. It is installed by typing in "**hsmgr -i**" at a command line.

Hotswap Monitor

The hotswap monitor test program is used to display the board level activity of the hotswap manager and driver. This program is located in the \hotswap\bin\intel directory. The Hotswap Manager (above) must be running for this to effectively work. It can be run at a command prompt by typing in "**hsmon**".

6.0 Source Code And Directory Structure

This package contains all of the source code for the XDS and hotswap device driver, DLLs, driver installation application, the hotswap manager service & monitor program, and test & communication programs. The following is a description of the directory hierarchy:

XDS Source Code Directories -

\XDS\bin\intel	- executables, DLLs, driver, and downloader
\XDS\source\Dlls\XdsLib110	- xdslib110.dll (XDS native functions)
\XDS\source\Dlls\XdsMv90	- xdsmv90.dll (MVIP-90 functions)
\XDS\source\Dlls\XdsMv95	- xdsmv95.dll (XDS functions)
\XDS\source\Dlls\XdsCtBus	- xdsctbus.dll (XDS functions)
\XDS\source\Downloader	- wn386dlc downloader program
\XDS\source\Driver	- xds_h110.sys low level driver
\XDS\source\Include	- include (header) files
\XDS\source\Lib	- library files for Intel x86 processors
\XDS\source\Wise	- WISE Installer project file
\XDS\source\Shared	- shared source code directory
\XDS\source\Sig_Util	- Sig_Util application source code
\XDS\source\Sig_Util\res	- Sig_Util project resource source code
\XDS\source\Signal_Test	- Signal_Test application source code
\XDS\source\Signal_Test\res	- Signal_Test project resource source code
\XDS\source\Test_dll	- xdslib110.dll test
\XDS\source\TestMv90	- xdsmv90.dll test
\XDS\source\TestMv95	- xdsmv95.dll test
\XDS\source\TestCtBus	- xdsctbus.dll test
\XDS\source\Test_drv	- test_drv.exe driver test
\XDS\source\Tstchs	- tstchs hot-plug test
\XDS\source\Xdsinst_h110	- xdsinst_h110 (installation) program
\XDS\source\Xdsutil	- xdsutil (utility) application

Hotswap Source Code Directories -

\hotswap\bin\intel	- executables, DLLs, and driver
\hotswap\source\Documents	- hotswap service and developer manuals
\hotswap\source\hslib	- hotswap library
\hotswap\source\hsdriver	- hotswap driver
\hotswap\source\hsmgr	- hotswap manager
\hotswap\source\hsmlib	- hotswap manager library
\hotswap\source\hsmon	- hotswap monitor
\hotswap\source\inc	- header files
\hotswap\source\inifile	- hotswap initialization file
\hotswap\source\lib	- hotswap libraries
\hotswap\source\tsilib	- tsi hotswap library
\hotswap\source\xdshslib	- XDS hotswap library

All the source subdirectories have a readme.txt file, which explains how the software can be rebuilt or recompiled.

This page was intentionally left blank.

XDS Windows NT
Driver IOCTL Description

This page was intentionally left blank.

Overview

The XDS Windows NT Driver is designed to provide an interface between XDS boards and applications running under Windows NT. It contains facilities to send and receive messages from any XDS board. There are also functions that allow for the direct reading and writing of the Dual-Ported Ram, which can be used for diagnostic and software downloading purposes.

A common interface is used by all XDS boards, regardless of type. Control of the boards is accomplished through command strings, which are in the form of NULL terminated ASCII strings that are in CAPTIOL letters. Responses, acknowledgments, state changes and error information are also passed from the XDS boards in the form of ASCII strings. Each board has a transmit and receive mailbox and a set of corresponding flags. Each board also provides a limited amount of buffering (eight messages deep) in either direction.

The application software should use the standard I/O functions CreateFile, CloseHandle, and DevIoControl to communicate with the device driver. The DevIoControl supports the following five commands:

XMT	- transmit a message to a board
RCV	- receive a message from the response queue
RCV_QUERY	- receive a message from the query queue
READ_DPRAM	- read from dual-ported RAM on a board
WRITE_DPRAM	- write to dual-ported RAM on a board
XDS_BOARD_ID	- obtain the board ID from DPRAM
XDS_RESET	- reset specified device (ISA High Density Line Boards, ISA BRI, all H.100, and all H.110 boards)
XDS_HR_ACK	- hardware removal (Hot-swap/H.110 only)
XDS_SLEEP	- hardware removal (Hot-plug/H.110 only)
XDS_/RESUME	- hardware addition (Hot-plug/H.110 only)

For the purposes of these commands, the board is specified by board_number.

For cPCI/H.110 boards, this number will correspond to the cPCI device number, given by the XDS device driver. These numbers will range from 1-30.

The transmit command writes messages directly to the mailbox of the appropriate board. The driver places received messages on one of two queues. Acknowledgments, state change messages, and error messages are passed through the receive queue. Query responses and Version Request responses are passed through a separate receive query queue. Each queue is shared by all of the XDS boards in the system. A driver command is provided for reading each queue. The receive queue can handle up to 31 messages while the query queue can handle 7. If the queue is full, the driver will discard additional messages. It is therefore the responsibility of the application to check the queues frequently enough so that they do not fill up.

The driver can be set to notify the application when a new message has arrived from an XDS board using the NT signaling mechanism. This facility eliminates the need for an application to continuously poll the driver.

Commands are provided for reading and writing the dual-ported RAM, which each board shares with the host processor. These commands include protection to prevent reading or writing outside of the dual ported memory on a particular board or for overwriting the mailboxes or configuration information on each board.

Application Interface

Applications can interface directly to the driver by using the CreateFile, CloseHandle, and DevIoControl function calls. Through the DevIoControl function, the application can send and receive messages directly to and from XDS boards. It is also possible to directly read or write to the Dual-Ported Ram on the XDS boards. OpenEventHandle is used to obtain an event handle for the signalling mechanism.

CreateFile
Before an application can access the DevIoControl function, a connection to the driver must be established and a file handle must be obtained. This function opens up a connection to the device driver. It returns a handle that is used to send requests to the device driver. All event queues will be initialized whenever a connection with the XDS driver is opened.

```
HANDLE CreateFile(LPCTSTR          lpFileName,
DWORD                              dwDesiredAccess,
DWORD                              dwShareMode,
LPSECURITY_ATTRIBUTES              lpSecurityAttributes,
DWORD                              dwCreationDistribution,
DWORD                              dwFlagsAndAttributes,
HANDLE                             hTemplateFile);
```

The XDS ISA/PCI H.100 device driver file name uses the symbolic link notation of "\\\\.\\XDS". The XDS cPCI H.110 device driver file name uses the symbolic link notation of "\\\\.\\XDS_h110". This symbolic link is set up when the device driver is installed.

CloseHandle
This function will close an open object handle returned by the CreateFile function.

```
BOOL CloseHandle(HANDLE hobject);
```

DevIoControl

The DevIoControl call takes the form:

BOOL DeviceIoControl (Handle hDevice,
DWORD dwIoControlCode,
LPVOID lpInBuffer,
DWORD nInBufferSize,
LPVOID lpOutBuffer,
DWORD nOutBufferSize,
LPDWORD lpBytesReturned,
LPOVERLAPPED lpOverlapped);

It sends the requested command code directly to the specified device driver. The driver will perform the operation and return a status flag indicating if the command was completed correctly.

All requests to the XDS device driver are made by calling this function. Each type of request may require different input and output structures, which are detailed in the following pages.

Several IOCTL commands are available to an application. They are:

XMT	- transmit a message to the board
RCV	- receive a message from the board
RCV_QUERY	- receive query response messages
READ_DPRAM	- read from dual-ported RAM
WRITE_DPRAM	- write to the dual-ported RAM
XDS_BOARD_ID	- obtain the board ID from DPRAM
XDS_RESET	- reset specified device (ISA High Density Line Boards, ISA BRI, all H.100, and all H.110 boards)
XDS_HR_ACK	- hardware removal (Hot-swap/H.110 only)

OpenEventHandle
This function is used to obtain the event handle for the signaling mechanism. The application makes a call to the function

OpenEventHandle(HANDLE, *hOut)

If this function succeeds, the event handle is stored in hOut and the function returns a 1. Otherwise, the event handle is null and the function returns a 0.
The application can then use the Win32 WaitForSingleObject call to wait on the event handle for incoming messages.

XMT

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XMT,	transmit message command
&msg,	pointer to message structure
sizeof(XDS_MSG),	length of message
NULL,	pointer to output structure
0,	length of output structure
&datalength,	pointer to number of bytes returned
NULL);	

XDS_MSG msg;	
typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose

This command is used to send messages to an XDS board. The board is specified in board_number in the structure **msg** which corresponds to the board number. The message is contained in the character array msg, and consists of a NULL terminated character string.

Returns

The function will return the following codes:

STATUS_SUCCESS	success
STATUS_DATA_ERROR	timeout or other problem with the board
STATUS_BUFFER_TOO_SMALL	insufficient memory allocated in call

Comments

Transmit messages are not queued, but sent directly to the board. If the mailbox is full, XDS_XMT will wait up to a tenth of a second before reporting a failure. Note that **augTxRxLen** and **augTxRxMesg** are only valid when sending a Layer 3 message to an XDS Basic Rate ISDN Board when the message in **msg** is of the format “LC” or “LR”.

RCV

```
BOOL DevIoControl(
hdriver,                    device handle
(DWORD)RCV,                 receive message command
NULL,                       pointer to input structure
0,                          length of input structure
&msg,                       pointer to output structure
sizeof(XDS_MSG),            length of output structure
&datalength,                pointer to number of bytes returned
NULL);

XDS_MSG     msg;

typedef struct{
UCHAR board_number;         the board number
CHAR msg[32];               the ASCII text of message, NULL terminated
USHORT augTxRxLen;          length of Layer 3 message
UCHAR augTxRxMesg[260];     body of Layer 3 message
}XDS_MSG *PXDS_MSG;
```

Purpose

This command is used to receive normal messages from XDS boards. Query and version request response messages are returned on the query response queue and read with the RCV_QUERY command. The board sending the message is contained in board_number, while the text of the message is in the character array msg in the form of a NULL terminated ASCII string.

Returns

The function will return the following codes:

STATUS_SUCCESS	success
STATUS_DATA_ERROR	no message available
STATUS_BUFFER_TOO_SMALL	insufficient memory allocated in call

Comments
This command checks to see if there is any message on the receive queue. If there is, it will return with the message. If no message is present, it will return immediately with a return value of STATUS_DATA_ERROR.

Normal messages are placed on the receive queue. These include acknowledgments, state change messages, and error messages. Version request and query responses are placed on the query response queue and can be read using the RCV_QUERY command.

The elements **augTxRxLen** and **augTxRxMesg** are only valid when receiving Layer 3 messages on the XDS Basic Rate ISDN Board and the message in **msg** is of the form "LC" or "LR".
If the queue becomes full, a "FULL QUEUE" message is placed on the queue with the board_number for that message set to 0xFF. If this message is received, it indicates the possibility that messages may have been lost. It is the responsibility of the application to check for messages often enough to prevent this.

RCV_QUERY

```
BOOL DevIoControl(
hdriver,                    device handle
(DWORD)RCV_QUERY,           receive query message command
NULL,                       pointer to input structure
0,                          length of input structure
&msg,                       pointer to output structure
sizeof(XDS_MSG),            length of output structure
&datalength,                pointer to number of bytes returned
NULL);

XDS_MSG     msg;

typedef struct{
UCHAR board_number;         the board number
CHAR msg[32];               the ASCII text of message, NULL terminated
USHORT augTxRxLen;          length of Layer 3 message
UCHAR augTxRxMesg[260];     body of Layer 3 message
}XDS_MSG *PXDS_MSG;
```

Purpose

This command is used to receive version request responses and query responses, which are placed on the query response queue by the driver. The board sending the message is contained in board_number, while the text of the message is in the character array msg as a NULL terminated ASCII string.

Returns

The function will return the following codes:

STATUS_SUCCESS	success
STATUS_DATA_ERROR	no message available
STATUS_BUFFER_TOO_SMALL	insufficient memory allocated in call

Comments
Unlike the RCV command, the RCV_QUERY command does not return immediately if there is no message available. It will wait up to a half of a second for a message to be placed on the queue. This implementation was made because of the finite time that it takes a board to respond to a version request or a query. By doing so, it eliminates the need for the application to implement a timeout mechanism.

Version response messages always begin with the letter 'V' and query responses always begin with the letter 'Q' or have 'Q' as the second letter and do not have a first letter of 'S' or 'E'. These messages are always placed on the query response queue and must be read using the RCV_QUERY command.

The elements **augTxRxLen** and **augTxRxMesg** never contain valid data when using RCV_QUERY.

If the queue becomes full, a "FULL QUEUE" message is placed on the queue with the board_number for that message set to 0xFF. If this message is received, it indicates the possibility that messages may have been lost. It is the responsibility of the application to check for messages often enough to prevent this.

READ_DPRAM

BOOL DevIoControl(
hdriver, device handle
(DWORD)READ_DPRAM, read from DPRAM command
&ram_info, pointer to input structure
sizeof(XDS_BOARD_RAM), length of input structure
NULL, pointer to output structure
0, length of output structure
&datalength, pointer to number of bytes returned
NULL);

XDS_BOARD_RAM ram_info;

typedef struct {
UCHAR board_number; the board number
ULONG offset; the offset in bytes into dual-ported RAM
ULONG size; the number of bytes to be read
UCHAR *buffer; pointer to the buffer to receive the bytes read
} XDS_BOARD_RAM *PXDS_BOARD_RAM

Purpose
This command can be used to read directly the contents of a portion of the dual-ported RAM. This may be done to obtain configuration information or for diagnostic purposes. The information read is placed in a buffer supplied by the application.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
STATUS_DATA_ERROR	attempt to read outside the on board RAM
STATUS_BUFFER_TOO_SMALL	insufficient memory allocated in call

Comments
This command may be used to obtain configuration information on the board, such as the board type, port states, etc. However, there also exist library functions that will accomplish the same results which may be easier to use. It is also possible to use this command for diagnostic purposes to display the contents of the mailboxes and the state of the transmit and receive flags.

WRITE_DPRAM

BOOL DevIoControl(

hdriver,	device handle
(DWORD)WRITE_DPRAM,	write to DPRAM command
&ram_info,	pointer to input structure
sizeof(XDS_BOARD_RAM),	length of input structure
NULL,	pointer to output structure
0,	length of output structure
&datalength,	pointer to number of bytes returned
NULL);	

XDS_BOARD_RAM ram_info;

typedef struct {

UCHAR board_number;	the board number
ULONG offset;	the offset in bytes into dual-ported RAM
ULONG size;	the number of bytes to be written
UCHAR *buffer;	pointer to the bytes to be written

} XDS_BOARD_RAM *PXDS_BOARD_RAM

Purpose

This command is used to write information into the dual-ported RAM on the XDS board specified in board_number. This is normally not necessary as the XMT command can be used to control the board. However, for diagnostic purposes, or for downloading firmware, this command may be used.

Returns

The function will return the following codes:

STATUS_SUCCESS	success
STATUS_DATA_ERROR	attempt to write to protected RAM or outside of on board RAM
STATUS_BUFFER_TOO_SMALL	insufficient memory allocated in call

Comments

The WRITE_DPRAM command is included in the command set to facilitate writing a firmware downloader. It normally will not be necessary for an application to use this command. It prevents writing to the first 256 bytes of the dual-ported RAM on ISA boards and the last 256 bytes on PCI boards. This area contains the mailboxes, flags, and configuration information for the board.

XDS_BOARD_ID

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_BOARD_ID,	board ID command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
pData,	pointer to output structure
sizeof(XDS_MSG),	length of output structure
&datalength,	pointer to number of bytes returned
NULL);	

XDS_MSG msg;

typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose
This command is used to obtain the ID of a specified board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
XDS_ERR_XDS_ID	error obtaining board ID, board may not be functioning properly

Comments
This function does not replace the xds_id() function in the XDS library. It will simply return the first two characters of the board ID.

XDS_RESET

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_RESET,	hardware reset command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
NULL,	pointer to output structure
0,	length of output structure
&datalength,	pointer to number of bytes returned
NULL);	

XDS_MSG msg;

typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose
This command is used to reset an entire board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
XDS_ERR_IOCTL_RESET	error resting board, board may not be functioning properly

Comments
This function does not replace the xds_reset_all() function in the XDS library. This will reset entire board. It is valid for the ISA High Density Boards, all ISA BRI boards, all PCI/H.100 boards, and all of the cPCI/H.110 boards.

XDS_HR_ACK

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_HR_ACK,	hardware removal acknowledge command
NULL,	pointer to input structure
0,	length of input structure
pData,	pointer to output structure
sizeof(XDS_MSG),	size of msg
&datalength,	pointer to number of bytes returned
NULL);	

XDS_MSG msg;

typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose
This command is used to monitor the removal of a cPCI/H.110 board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
XDS_ERR_IOCTL_RESET	error removing board from the IOCTL

Comments
This function is used only with XDS cPCI/H.110 hot-swap boards. It is a useful command when using the hot-swap facilities of the hot-swap driver.

XDS_SLEEP

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_SLEEP,	board removal command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
NULL	not used
0,	not used
&datalength,	pointer to number of bytes returned
NULL);	not used

XDS_MSG msg;

typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose
This command is used to remove a cPCI/H.110 board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
XDS_ERR_IOCTL_SLEEP	error removing board

Comments
This function is used only with XDS cPCI/H.110 hot-swap boards. It is a useful command when using the hot-plug capabilities of the XDS driver.

XDS_RESUME

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_RESUME,	board insert command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
NULL	not used
0,	not used
&datalength,	pointer to number of bytes returned
NULL);	not used

XDS_MSG msg;

typedef struct{	
UCHAR board_number;	the board number
CHAR msg[32];	the ASCII text of message, NULL terminated
USHORT augTxRxLen;	length of Layer 3 message
UCHAR augTxRxMesg[260];	body of Layer 3 message
}XDS_MSG *PXDS_MSG;	

Purpose
This command is used to insert a cPCI/H.110 board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
XDS_ERR_IOCTL_SLEEP	error removing board

Comments
This function is used only with XDS cPCI/H.110 hot-swap boards. It is a useful command when using the hot-plug capabilities of the XDS driver.

This page was intentionally left blank.

MVIP-90 Software Interface Description

This page was intentionally left blank.

MVIP-90 Software Standard
The MVIP-90 Software Standard provides a uniform interface for MVIP boards. The standard specifies a set of commands and responses for controlling switching and system clocks. Vendor specific commands may be added to this set as necessary as long as these commands conform to the rules of the specification. These commands may be necessary to control board functions that are outside of the scope of the MVIP-90 Standard.

Windows NT/2000 Implementation
The specific implementation for Windows NT/2000 is as a dynamic link library (DLL). The library must export a single entry point called **SwDevIOCtl().** This DLL may perform hardware I/O operations directly or may serve as the interface to a Windows NT/2000 device driver. For the XDS MVIP driver, the latter method is used using the driver described in the previous section.
The DLL function declaration is:

INT SWDEVIOCTL(INT device_number, INT cmd, INT* p)

The application interface to the DLL is:

module_handle = LoadLibrary(DLL_name);
mvipIOCtl = GetProcAddress(module_handle, “SWDEVIOCTL”);
rc = mvipIOCtl(device_number, cmd, &p);

where:
(HINSTANCE) module_handle is the Windows NT reference to the DLL module.
(FARPROC) mvipIOCtl is the Windows NT reference to the DLL entry point function
(INT) device_number is a specific switch block number
(INT) cmd is the command code represented
(INT *) p is the command’s parameter, usually a pointer to a structure.
(INT) rc is the MVIP error code.

For the XDS MVIP Driver, the device_number will correspond to the SW1 setting of an ISA board or PCI device number of the board for which the command is being issued. The DLL is named **XDSMV90.DLL**.

Parameters

Parameters for the various commands are usually passed in a structure. The **ioctl** call contains a pointer to this structure. Because of differences between commands, the parameter structure varies from command to command. These structures are documented in the command reference sections.

Error Codes

Windows NT does not return error codes directly from DeviceIoControl. Rather TRUE or FALSE are returned and the GetLastError function is used to determine what error occurred. The DLL is responsible for extracting this information and translating it in an appropriate manner. Error codes returned by the DLL fall into three categories: general device errors, parameter value errors, and switching related errors. Code 0, which is SUCCESS, and codes 200 through 229 are specified as part of the MVIP-90 Standard. Other codes, above a certain number, are available for vendor specific use. The error codes are listed in a table in the "MVIP-Related Error Codes" chapter.

XDS MVIP Driver Command Set

The XDS MVIP Driver implements all of the mandatory commands in the MVIP-90 Standard. In addition, XDS specific commands are included for controlling the XDS MVIP Multi-Chassis Board, the XDS Switch Matrix Board, and the XDS MVIP Line Interface Boards (DID, E&M, Ground Start, Loop Start and Station Boards). These commands are grouped in four subsets described in the following sections: Generic XDS Commands, MVIP Commands, Multi-Chassis and Switch Matrix Commands, and Line Interface Commands. The command codes are listed in a table at the end of this document.

Generic Commands
These are commands that work with all XDS boards. Included in this set are commands to reset the boards, request board identification information, enable messages from the board and set the encoding format of audio signals to A-Law or Mu-Law. In addition, there are commands to send native mode messages to the boards and to receive messages from the board.

MVIP-90 Commands
This is the set of mandatory commands specified in the MVIP-90 Standard. These commands are for controlling the clocks and switching as well as diagnostics. The exact implementation of these commands may vary depending on the board type.

Multi-Chassis & Switch Matrix Commands
Included in this set of commands are the commands to control the MC1 Multi-Chassis Interface bus and the clocks associated with it. In addition, there is a command to implement conferencing on both the Multi-Chassis and Switch Matrix Board. Also, there are commands to access the DSP resources on the Switch Matrix and to configure the MVIP interface on that board.

Line Board Commands
These commands are used to control the analog line interface circuits on the XDS MVIP DID, E&M, Ground Start, Loop Start and Station Boards as well as B-channel control of the XDS MVIP Basic Rate ISDN Boards. Included are commands to configure these ports and to seize and release the lines associated with them. There are also commands to send and receive DTMF signals, send call progress signals and generate hook-flashes. Commands specific to the Station board can generate ringing and control the message waiting indicator.

This page was intentionally left blank.

MVIP-95 Software Interface Description

This page was intentionally left blank.

MVIP-95 BUS Software Standard

The **MVIP-95** Software Standard provides a uniform interface for MVIP, H.100, and H.110 boards. The standard specifies a set of commands and responses for controlling switching and system clocks. Vendor specific commands may be added to this set as necessary as long as these commands conform to the rules of the specification. These commands may be necessary to control board functions that are outside of the scope of the MVIP-95 Standard.

Windows NT/2000 Implementation

The specific implementation for Windows NT/2000 is as a dynamic link library (DLL). The library must export a single entry point called **SwDevIOCtl().** This DLL may perform hardware I/O operations directly or may serve as the interface to a Windows NT/2000 device driver. For the XDS driver, the latter method is used using the driver described in the previous section.
The DLL function declaration is:

INT SWDEVIOCTL(INT device_number, INT cmd, INT* p)

The application interface to the DLL is:

module_handle = LoadLibrary(DLL_name);
swdevioctl = GetProcAddress(module_handle, “SWDEVIOCTL”);
rc = swdevioctl(device_number, cmd, &p);

where:
(HINSTANCE) module_handle is the Windows NT reference to the DLL module.
(FARPROC)swdevioctl is the Windows NT reference to the DLL entry point function
(INT) device_number is a specific switch block number
(INT) cmd is the command code represented
(INT *) p is the command’s parameter, usually a pointer to a structure.
(INT) rc is the error code.

For the XDS Driver, the device_number will correspond to the SW1 setting of an

ISA board or the PCI device number of the board for which the command is being issued. The DLL is named **XDSMV95.DLL**.

Parameters
Parameters for the various commands are usually passed in a structure. The **ioctl** call contains a pointer to this structure. Because of differences between commands, the parameter structure varies from command to command. These structures are documented in the command reference sections.

Error Codes
Windows NT does not return error codes directly from DeviceIoControl. Rather TRUE or FALSE are returned and the GetLastError function is used to determine what error occurred. The DLL is responsible for extracting this information and translating it in an appropriate manner. Error codes returned by the DLL fall into three categories: general device errors, parameter value errors, and switching related errors. Code 0, which is SUCCESS, and codes 200 through 229 are specified as part of the MVIP-95 Standard. Other codes, above a certain number, are available for vendor specific use. The error codes are listed in a table in the "MVIP-Related Error Codes" chapter.

XDS MVIP-95 Driver Command Set
The XDS Driver implements all of the mandatory commands in the MVIP-95 Standard. In addition, XDS specific commands are included for controlling the XDS MVIP Multi-Chassis Boards, the XDS Switch Matrix Board, the XDS MVIP Line Interface Boards (DID, E&M, Ground Start, Loop Start and Station Boards), and the XDS BRI Interface Boards. These commands are grouped in four subsets described in the following sections: Generic XDS Commands, MVIP-95 Commands, Multi-Chassis and Switch Matrix Commands, and Line Interface Commands. The command codes are listed in a table at the end of this document.

Generic Commands
These are commands that work with all XDS boards. Included in this set are commands to reset the boards, request board identification information, enable messages from the board and set the encoding format of audio signals to A-Law or Mu-Law. In addition, there are commands to send native mode messages to the boards and to receive messages from the board.

Multi-Chassis & Switch Matrix Commands

Included in this set of commands are the commands to control the MC1 Multi-Chassis Interface bus and the clocks associated with it. In addition, there is a command to implement conferencing on both the Multi-Chassis and Switch Matrix Board. Also, there are commands to access the DSP resources on the Switch Matrix and to configure the MVIP interface on that board.

Line Board Commands

These commands are used to control the analog line interface circuits on the XDS MVIP DID, E&M, Ground Start, Loop Start and Station Boards as well as B-channel control of the XDS MVIP Basic Rate ISDN Boards. Included are commands to configure these ports and to seize and release the lines associated with them. There are also commands to send and receive DTMF signals, send call progress signals and generate hook-flashes. Commands specific to the Station board can generate ringing and control the message waiting indicator.

This page was intentionally left blank.

XDS MVIP-95
Command Reference

This page was intentionally left blank.

MVIP95_CMD_CONFIG_8KREF_CLOCK

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_mc1_8kref_clock_parms

```
struct mvip95_config_mc1_8kref_clock_parms {
int size;                 specifies size of the struct used
int clock_source;         specifies the clock reference from
int network               which network, if clock_source == MVIP95_SOURCE_NETWORK
};
```

Applicable Boards
XDS MC1 Multi-Chassis Boards

Purpose
This command configures the source of the MC1 8KREF signal. The source can be an internal oscillator, the MVIP bus clocks, or no source.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_CLOCK_PARM

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a "SUCCESS" message and the clock mode will remain unchanged.

MVIP95_CMD_CONFIG_BOARD_CLOCK

device number: the device handle for the XDS board to receive the command
parameters:

&mvip95_config_mc1_board_clock_parms (MC1 Boards)

struct mvip95_config_mc1_board_clock_parms {

int size;	specifies size of the struct used
int clock_type;	indicates the MVIP standard clocking used on the board
int clock_source;	specifies where the clock reference originates
int network;	the device source for the clock signals (if source ==network)
int mc1_clock_mode;	specifies the board's control of the MC1 clocks
int auto_fall_back;	specifies whether the board is to automatically switch to the fall back mode and become a slave to alternate MC1 clock
int fall_back_occurred;	specifies whether the board has detected the primary master clock signal has become unreliable and fallen back to a secondary source

&mvip95_config_h100_board_clock_parms (H.100/110 Boards)

struct mvip95_cinfig_h100_board_clock_parms {

int size;	specifies size of the struct used
int clock_type;	indicates the MVIP standard clocking used on the board
int clock_source;	specifies where the clock reference originates
int network;	the device source for the clock signals (if source == network)
int mc1_clock_mode;	specifies the board's control of the MC1 clocks
int auto_fall_back;	specifies whether the board is to automatically switch to the fall back mode and become a slave to alternate MC1 clock
int netref_clock_speed;	specifies speed of the NETREF clock signal

&mvip95_config_hmvip_board_clock_parms (All other MVIP Boards)

struct mvip95_config_hmvip_board_clock_parms {

int size;	specifies size of the struct used
int clock_type;	indicates the MVIP standard clocking used on the board
int clock_source;	specifies where the clock reference originates
int network;	the device source for the clock signals (if source == network)

Applicable Boards

All XDS boards.

Purpose
This command configures selected board to all of the MVIP95 requirements specified.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_CLOCK_PARM
MVIP95_ERR_INVALID_PARAMETER

Message Sent

MC1:	"SCxx"- where **xx** is the clock mode
H.100/110:	"SCmsabb(c)"- where **m** is the clock mode, **s** is the sub-mode, **a** is the CT_NETREF, **bb** will be the reference frequency for submodes 1&2, **bb** will be the local network for submodes 3 – 5, and **c** will select the reference frequency of the CT_NETREF fallback source for sub-modes 4 & 5.

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a "SUCCESS" message and the clock mode will remain unchanged.

MVIP95_CMD_CONFIG_LOCAL_STREAM

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_local_stream_parms

```
struct mvip95_config_local_stream_parms {
int size;                 specifies size of the struct used
int local_stream;         the selected stream on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command returns information about the switch and its capabilities.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_PARM
MVIP95_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will return **MVIP95_ERR_NOT_CONFIGURABLE**.

MVIP95_CMD_CONFIG_LOCAL_TIMESLOT

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_local_timeslot_parms

```
struct mvip95_config_local_timeslot_parms {
int size;                 specifies size of the struct used
int local_stream;         the selected stream on local bus
int local_timeslot;       the selected timeslot on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command returns information about the switch and its capabilities.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_PARM
MVIP95_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will return **MVIP95_ERR_NOT_CONFIGURABLE**.

MVIP95_CMD_CONFIG_NETREF_CLOCK

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_netref_clock_parms

```
struct mvip95_config_netref_clock_parms {
int size;                   specifies size of the struct used
int network                 which network
int netref_clock_mode       board's control of secondary network clocks
int netref_clock_speed      which network (if clock_source == MVIP95_SOURCE_NETWORK)
};
```

Applicable Boards
XDS H.100 and H.110 boards.

Purpose
This command defines the secondary network reference clocks.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_CLOCK_PARM
MVIP95_ERR_INVALID_PARAMETER

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Only available clock speed for our boards is 8 KHz.

MVIP95_CMD_CONFIG_SEC8K_CLOCK

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_sec8k_clock_parms

```
struct mc1_sec8k_parms {
int clock_source;        specifies the clock reference from:
int network              which network if clock_source == MVIP95_SOURCE_NETWORK
};
```

Applicable Boards
All XDS boards.

Purpose
This command defines the secondary 8KHz - the network device from which SEC8K is obtained.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_CLOCK_PARM
MVIP95_ERR_INVALID_PARMAMETER

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a "SUCCESS" message and the clock mode will remain unchanged.

MVIP95_CMD_CONFIG_STREAM_SPEED

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_config_stream_speed_parms

```
struct mvip95_query_stream_speed_parms  {
int size;                   specifies size of the struct used
int speed;                  specifies the speed of the specified stream(s)
int *stream;                specifies the stream(s) selected for configuring
};
```

Applicable Boards
XDS H.100 boards

Purpose
This configures the stream speeds on a CT Bus.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_SPEED
MVIP95_ERR_INVALID_STREAM
MVIP95_ERR_INVALID_PARMAMETER

Message Sent
"SBabcd" where **a**, **b**, **c**, and **d** are blocks of 4 streams each on the CT bus.

Response
None

Comments
This command configures the selected streams for the selected speed(s). Only the lower 16 streams are configurable on the CT bus.

MVIP95_CMD_QUERY_BOARD_CLOCK

device number: the device handle for the XDS board to receive the command
parameters:

&mvip95_query_mc1_board_clock_parms (MC1 Boards)

```
struct mvip95_query_mc1_board_clock_parms {
int size;                   specifies size of the struct used
int clock_type;             indicates the MVIP standard clocking used on the board
int clock_source;           specifies where the clock reference originates
int network;                the device source for the clock signals (if source == network)
int mc1_clock_mode;         specifies the board's control of the MC1 clocks
int auto_fall_back;         specifies whether the board is to automatically switch to the fall
                            back mode and become a slave to alternate MC1 clock
int fall_back_occurred;     specifies whether the board has detected the primary master clock
                            signal has become unreliable and fallen back to a secondary source
}
```

&mvip95_query_h100_board_clock_parms (H.100/110 Boards)

```
struct mvip95_query_h100_board_clock_parms {
int size;                   specifies size of the struct used
int clock_type;             indicates the MVIP standard clocking used on the board
int clock_source;           specifies where the clock reference originates
int network;                the device source for the clock signals (if source == network)
int mc1_clock_mode;         specifies the board's control of the MC1 clocks
int auto_fall_back;         specifies whether the board is to automatically switch to the fall
                            back mode and become a slave to alternate MC1 clock
int fall_back_occurred;     specifies whether the board has detected the primary master clock
                            signal has become unreliable and fallen back to a secondary source
int h100_a_clock_status;    reports quality/status of the 'A' clock master signal
int h100_b_clock_status;    reports quality/status of the 'B' clock master signal
int netref_a_clock_status;  reports quality/status of the NETREF_A clock secondary signal
int netref_b_clock_status;  reports quality/status of the NETREF_B clock secondary signal
}
```

&mvip95_query_hmvip_board_clock_parms (All other MVIP Boards)

```
struct mvip95_query_hmvip_board_clock_parms {
int size;                   specifies size of the struct used
int clock_type;             indicates the MVIP standard clocking used on the board
int clock_source;           specifies where the clock reference originates
int network;                the device source for the clock signals (if source == network)
}
```

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the board's clock modes.

Returns
MVIP95_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the current clock mode of the specified board. If **config_8kref_clock** and/or **config_sec8k_clock** are called before this function, this function will return "SUCCESS" and do nothing.

MVIP95_CMD_QUERY_BOARD_INFO

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_board_info_parms

```
struct mvip95_query_board_info_parms {
int size;                   specifies size of the struct used
int description[80];        receives the device driver description
int revision[16];           receives the revision level of device driver
int date[12];               release date of the device driver
int vendor[80];             receives the name of the vendor of the device driver
int serial_number[80];      receives the serial number of a specified board
int board_id;               receives the vendor-specific identity number
int base_port_address       receives the physical I/O address of board
}
```

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the board.

Returns
MVIP95_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the selected hardware. The serial_number field will always be “N/A”, no XDS boards have electronically embedded serial numbers. The date will always be 0000/00/00, again, no XDS boards have embedded dates. The base_port_address will always be 0xFFFFF, because of limitations of reading the hardware.

MVIP95_CMD_QUERY_DRIVER_INFO

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_driver_info_parms

struct mvip95_query_driver_info_parms {

int size;	specifies size of the struct used
int description[80];	receives the device driver description
int revision[16];	receives the revision level of device driver
int date[12];	release date of the device driver
int vendor[80];	receives the name of the vendor of the device driver

}

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the driver.

Returns
MVIP95_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the device driver. The date will always be 0000/00/00, no XDS boards have electronically embedded dates.

MVIP95_CMD_QUERY_LOCAL_STREAM

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_local_stream_parms

```
struct mvip95_query_local_stream_parms {
int size;                 specifies size of the struct used
int local_stream;         the selected stream on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command is not supported by XDS boards.

Returns
MVIP95_SUCCESS
MVIP95_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will always return **MVIP95_ERR_NOT_CONFIGURABLE**.

MVIP95_CMD_QUERY_LOCAL_TIMESLOT

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_local_timeslot_parms

```
struct mvip95_query_local_timeslot_parms {
int size;                 specifies size of the struct used
int local_stream;         the selected stream on local bus
int local_timeslot;       the selected timeslot on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command is not supported by XDS boards.

Returns
MVIP95_SUCCESS
MVIP95_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will always return **MVIP95_ERR_NOT_CONFIGURABLE**.

MVIP95_CMD_QUERY_OUTPUT

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_output_parms

```
struct mvip95_query_output_parms {
int size;                       specifies size of the struct used
MVIP95_OUTDESC *output;         specifies the switch block outputs
};
```

Applicable Boards
All XDS boards.

Purpose
This command retrieves output information on a terminus.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_STREAM
MVIP95_ERR_INVALID_TIMESLOT
MVIP95_ERR_INVALID_MODE
MVIP95_ERR_INVALID_PARAMETER

Message Sent
None

Response
None

Comments
For all the XDS MVIP boards, this command interrogates tables to obtain the information. For MVIP streams, a single table is kept for all boards. For local streams including conferences and the MC1 bus, the driver checks the relevant table to return information on whether a timeslot is active or not, and what timeslot is the input or pattern is being output.

MVIP95_CMD_QUERY_STREAM_SPEED

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_stream_speed_parms

```
struct mvip95_query_stream_speed_parms  {
int size;                       specifies size of the struct used
int speed;                      specifies the speed of the specified stream
int *stream;                    specifies the stream(s) selected for query
};
```

Applicable Boards
XDS H.100 boards.

Purpose
This command retrieves the speed of a specific stream.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_SPEED
MVIP95_ERR_INVALID_PARM

Message Sent
None

Response
None

Comments
This command reads dual-ported RAM for query information. Because of hardware limitations, streams are configured in blocks of four each (0-3, 4-7, 8-11, 12-15). So, this function will return the stream speed of each block, not an actual stream.

Example
When querying speed for stream 0, it will specify the speed for the first block (0-3). In addition, the MVIP95 specification limits the “speed” parameter to only one value, so when querying blocks that may have different speeds, this function may be called several times.

MVIP95_CMD_QUERY_SWITCH_CAPS

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_query_switch_caps_parms

struct mvip95_query_switch_caps_parms {
int size; specifies size of the struct used
int dvr_ revision; receives the revision level of the device driver (multiplied by 100)
int domain; receives the domain of the switch block
int routing; receives switch block's half duplex routing capabilities
int blocking; receives switch block's possible blocking
int sw_standard; the MVIP software standard being used
int sw_std_revision; the revision of the MVIP software standard being used
int hw_standard; the MVIP standard being used
int hw_std_revision; the revision of the driver being used (multiplied by 100)
MVIP95_LOCAL_DEVICE_DESC
*local_devs; a pointer receiving the number of timeslots
and device type of each local stream
}

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the switch and its capabilities.

Returns
MVIP95_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the switching capabilities of the board. Note that the information is hard coded into the driver and is not returned by the board.

MVIP95_CMD_RESET_SWITCH

device number: the device handle for the XDS board to receive the command
parameters: none

Applicable Boards
All XDS boards

Purpose
This function can be used to put a board in a known, initialized state. All ports are released, all connections are broken, and all resources are freed. Outputs to the MVIP bus are disabled. This command does not change the clock mode of the board.

Returns
MVIP95_SUCCESS

Message Sent
"RA"

Response
All XDS boards respond with a message of type 2, subtype 1. The Switch Matrix Board makes no response.

Comments
This function should be used for all XDS boards when starting an application to put the boards in a known state. All connections are dropped and all resources are freed. The clock mode of the board is not altered by this command.

MVIP95_CMD_SAMPLE_INPUT

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_sample_input_parms

```
struct mvip95_sample_input_parms {
int size;                    specifies size of the struct used
MVIP95_INDESC *input;        specifies the switch block inputs
};
```

Applicable Boards
All XDS Legacy/ISA boards, except the Switch Matrix board.

Purpose
This command retrieves the currently asserted byte on a switch block input.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_TIMESLOT
MVIP95_ERR_INVALID_STREAM
MVIP95_ERR_INVALID_PARAMETER

Message Sent
"QIsstt"

Response
None

Comments
This command causes the board to read the data memory of the FMIC chip to find the value asserted. In the case of the Multi-Chassis board, the board uses FMIC 2 to read the information if the stream is less than 0x13. Streams 0x10-0x13 are the local streams used by FMIC 2 to connect to the conference chips. If the stream number is greater than or equal to 0x14 the board will look for a connection on the MC1 bus for that stream and timeslot. If there is no such connection, then a value of 0xFF will be returned as the sample value. The Switch Matrix Board does not have an FMIC and does not support this command.

MVIP95_CMD_SET_OUTPUT

device number: the device handle for the XDS board to receive the command
parameters: &mvip95_set_output_parms

```
struct mvip95_set_output_parms {
int size;                          specifies size of the struct used
MVIP95_OUTDESC *output;            specifies the switch block outputs
};
```

Applicable Boards
All XDS boards.

Purpose
This command is used to make and break connections, to disable a switch block output, or optionally, to continuously output a fixed pattern on a switch block output.

Returns
MVIP95_SUCCESS
MVIP95_ERR_INVALID_TIMESLOT
MVIP95_ERR_INVALID_STREAM
MVIP95_ERR_INVALID_PARAMETER

Message Sent
"MOssttiiiimpp" for Line boards and BRI boards
"SOssttiiiimpp" for the Multi-chassis board
where **sstt** is the output stream and timeslot, **iiii** is the input stream and timeslot, **m** is the mode and **pp** is the pattern value
"CLxxxyyy" for the Switch Matrix board in the connect mode
"CDxxx" for the Switch Matrix board in the disable mode where **xxx** is the output stream and timeslot and **yyy** is the input stream and timeslot

Response
None

Comments

The **MVIP95_CMD_SET_OUTPUT** command can be used to create connections using any of the switch blocks on the MC1 Multi-Chassis board. Streams 0x00-0x0F are the MVIP streams. Streams 0x10-0x13 are the local streams used to connect to the conferencing hardware. Streams 0x14-0x2B are the MC1 streams. Note, that to conference, additional commands must be issued to the board. A maximum of four streams may be used for transmitting to the MC1 bus. The messages to the board reflect this in that only streams numbered 0x14-0x17 are used. The library makes a translation from the range 0x14-0x2B to this range.

For the XDS Line boards, the **MVIP95_CMD_SET_OUTPUT** command controls the FMIC. It does not control either the seize function or the CODEC function of each port. To create a connection, an **XDS_MVIP_CONNECT** command must also be issued. The order of these commands is not important to the functioning of the board. To release a port, the **XDS_RLS** command must be used.

As the Switch Matrix board does not use an FMIC as the switch block, the actions of a **MVIP95_CMD_SET_OUTPUT** are approximated with the listen and disconnect messages to the board. There is no pattern capability on the Switch Matrix board. The DLL translates the streams in the **MVIP95_CMD_SET_OUTPUT** command to the appropriate values for the CL and CD commands used by the board. MVIP streams 0x0-0xF will map to streams 8-F on the board depending on the parameters sent to the **XDS_MX_SET_DIRECTION** command. MVIP streams 0x10-0x17 become 0-7 on the board. Streams 0-6 refer to the APIB connectors. Stream 7 is the PEB connector. Stream 6 may also be used to connect to the on-board DSPs.

This page was intentionally left blank.

CT-BUS Software Interface Description

This page was intentionally left blank.

CT-BUS BUS Software Standard
The **CT-BUS** Software Standard provides a uniform interface for MVIP, H.100, and H.110 boards. The standard specifies a set of commands and responses for controlling switching and system clocks. Vendor specific commands may be added to this set as necessary as long as these commands conform to the rules of the specification. These commands may be necessary to control board functions that are outside of the scope of the CT-BUS Standard.

Windows NT/2000 Implementation
The specific implementation for Windows NT/2000 is as a dynamic link library (DLL). The library must export a single entry point called **SwDevIOCtl().** This DLL may perform hardware I/O operations directly or may serve as the interface to a Windows NT/2000 device driver. For the XDS driver, the latter method is used using the driver described in the previous section.
The DLL function declaration is:

INT SWDEVIOCTL(INT device_number, INT cmd, INT* p)

The application interface to the DLL is:

module_handle = LoadLibrary(DLL_name);
swdevioctl = GetProcAddress(module_handle, “SWDEVIOCTL”);
rc = swdevioctl(device_number, cmd, &p);

where:
(HINSTANCE) module_handle is the Windows NT reference to the DLL module.
(FARPROC)swdevioctl is the Windows NT reference to the DLL entry point function
(INT) device_number is a specific switch block number
(INT) cmd is the command code represented
(INT *) p is the command’s parameter, usually a pointer to a structure.
(INT) rc is the error code.

For the XDS Driver, the device_number will correspond to the SW1 setting on an ISA board or the PCI device number of the board for which the command is being issued. The DLL is named **XdsCtBus.DLL**.

Parameters

Parameters for the various commands are usually passed in a structure. The **ioctl** call contains a pointer to this structure. Because of differences between commands, the parameter structure varies from command to command. These structures are documented in the command reference sections.

Error Codes

Windows NT does not return error codes directly from DeviceIoControl. Rather TRUE or FALSE are returned and the GetLastError function is used to determine what error occurred. The DLL is responsible for extracting this information and translating it in an appropriate manner. Error codes returned by the DLL fall into three categories: general device errors, parameter value errors, and switching related errors. Code 0, which is SUCCESS, and codes 200 through 229 are specified as part of the CT-BUS Standard. Other codes, above a certain number, are available for vendor specific use. The error codes are listed in a table in the "MVIP-Related Error Codes" chapter.

XDS CT-BUS Driver Command Set

The XDS Driver implements all of the mandatory commands in the CT-BUS Standard. In addition, XDS specific commands are included for controlling the XDS MVIP Multi-Chassis Boards, the XDS Switch Matrix Board, the XDS MVIP Line Interface Boards (DID, E&M, Ground Start, Loop Start and Station Boards), and the XDS BRI Interface Boards. These commands are grouped in four subsets described in the following sections: Generic XDS Commands, CT-BUS Commands, Multi-Chassis and Switch Matrix Commands, and Line Interface Commands. The command codes are listed in a table at the end of this document.

Generic Commands
These are commands that work with all XDS boards. Included in this set are commands to reset the boards, request board identification information, enable messages from the board and set the encoding format of audio signals to A-Law or Mu-Law. In addition, there are commands to send native mode messages to the boards and to receive messages from the board.

Multi-Chassis & Switch Matrix Commands
Included in this set of commands are the commands to control the MC1 Multi-Chassis Interface bus and the clocks associated with it. In addition, there is a command to implement conferencing on both the Multi-Chassis and Switch Matrix Board. Also, there are commands to access the DSP resources on the Switch Matrix and to configure the MVIP interface on that board.

Line Board Commands
These commands are used to control the analog line interface circuits on the XDS MVIP DID, E&M, Ground Start, Loop Start and Station Boards as well as B-channel control of the XDS MVIP Basic Rate ISDN Boards. Included are commands to configure these ports and to seize and release the lines associated with them. There are also commands to send and receive DTMF signals, send call progress signals and generate hook-flashes. Commands specific to the Station board can generate ringing and control the message waiting indicator.

This page was intentionally left blank.

XDS MVIP CT-BUS
Command Reference

This page was intentionally left blank.

CTBUS_CMD_CONFIG_8KREF_CLOCK

command: CTBUS_CMD_CONFIG_8KREF_CLOCK
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_mc1_8kref_clock_parms

```
struct ctbus_config_mc1_8kref_clock_parms {
int size;              specifies size of struct used
int clock_source;      specifies the clock reference from:
int network            which network, if clock_source == CTBUS_SOURCE_NETWORK
};
```

Applicable Boards
XDS MC1 Multi-Chassis Boards

Purpose
This command configures the source of the MC1 8KREF signal. The source can be an internal oscillator, the MVIP bus clocks, or no source.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_CLOCK_PARM

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a "SUCCESS" message and the clock mode will remain unchanged.

CTBUS_CMD_CONFIG_BOARD_CLOCK

command: CTBUS_CMD_CONFIG_BOARD_CLOCK
device number: the device handle for the XDS board to receive the command
parameters:
&ctbus_config_h100_board_clock_parms (H.100/110 Boards)
struct ctbus_cinfig_h100_board_clock_parms {

int size;	specifies size of struct used
int clock_type;	indicates the MVIP standard clocking used on the board
int clock_source;	specifies where the clock reference originates
int network;	the device source for the clock signals (if source == network)
int h100_clock_mode;	specifies the board's control of the H100 clocks
int auto_fall_back;	specifies whether the board is to automatically switch to the fall back mode and become a slave to alternate MC1 clock
int netref_clock_speed;	specifies speed of the NETREF clock signal
int fall_back_clock_source;	specifies the source of the clock when fall back occurs
int fall_back_network;	specifies the on-board source for the network fall back clock

Applicable Boards
All XDS boards.

Purpose
This command configures selected board to all of the CTBUS requirements specified.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_PARM
CTBUS_ERR_INVALID_PARAMETER

Message Sent

MC1:	"SCxx"- where **xx** is the clock mode
H.100/110:	"SCmsabb(c)"- where **m** is the clock mode, **s** is the sub-mode, **a** is the CT_NETREF, **bb** will be the reference frequency for submodes 1&2, **bb** will be the local network for submodes 3 – 5, and **c** will select the reference frequency of the CT_NETREF fallback source for sub-modes 4 & 5.

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a “SUCCESS” message and the clock mode will remain unchanged.

CTBUS_CMD_CONFIG_LOCAL_STREAM

command: CTBUS_CMD_CONFIG_LOCAL_STREAM
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_local_stream_parms

```
struct ctbus_config_local_stream_parms {
int size;                 specifies size of struct used
int local_stream;         the selected stream on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This configures the stream speeds on a CT Bus.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_PARM
CTBUS_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will return **CTBUS_ERR_NOT_CONFIGURABLE**.

CTBUS_CMD_CONFIG_LOCAL_TIMESLOT

command: CTBUS_CMD_CONFIG_LOCAL_TIMESLOT
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_local_timeslot_parms

```
struct ctbus_config_local_timeslot_parms {
int size;                 specifies size of struct used
int local_stream;         the selected stream on local bus
int local_timeslot;       the selected timeslot on local bus
int device_id;            device type on stream and timeslot selected
int parameter_id;         data item for configuration information obtained
int *buffer;              timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command returns information about the switch and its capabilities.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_PARM
CTBUS_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will return **CTBUS_ERR_NOT_CONFIGURABLE**.

CTBUS_CMD_CONFIG_NETREF_CLOCK

command: CTBUS_CMD_CONFIG_NETREF_CLOCK
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_netref_clock_parms

```
struct ctbus_config_netref_clock_parms {
int size;                   specifies size of struct used
int network                 which network
int netref_clock_mode       board's control of secondary network clocks
int netref_clock_speed      which network (if clock_source == CTBUS_SOURCE_NETWORK)
};
```

Applicable Boards
XDS H.100 and H.110 boards.

Purpose
This command defines the secondary network reference clocks.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_CLOCK_PARM
CTBUS_ERR_INVALID_PARAMETER

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Only available clock speed for our boards is 8 KHz.

CTBUS_CMD_CONFIG_SEC8K_CLOCK

command: CTBUS_CMD_CONFIG_SEC8K_CLOCK
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_sec8k_clock_parms

```
struct mc1_sec8k_parms {
int clock_source;          specifies the clock reference from:
int network                which network (if clock_source == CTBUS_SOURCE_NETWORK)
};
```

Applicable Boards
All XDS boards.

Purpose
This command defines the secondary 8KHz - the network device from which SEC8K is obtained.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_CLOCK_PARM
CTBUS_ERR_INVALID_PARMAMETER

Message Sent
"SCxx" where **xx** is the clock mode

Response
None

Comments
Because of the complexities of the clocking modes on the Multi-Chassis boards it is possible for other commands to put a board in a conflicting mode, such as SEC8K or 8KREF. If this is the case, the board will return a "SUCCESS" message and the clock mode will remain unchanged.

CTBUS_CMD_CONFIG_STREAM_SPEED

command: CTBUS_CMD_CONFIG_STREAM_SPEED
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_config_stream_speed_parms

```
struct ctbus_query_stream_speed_parms  {
int size;                    specifies size of struct used
int speed;                   specifies the speed of the specified stream
int *stream;                 specifies the stream(s) selected to be configured
};
```

Applicable Boards
XDS H.100 boards

Purpose
This configures the stream speeds on a CT Bus.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_SPEED
CTBUS_ERR_INVALID_STREAM
CTBUS_ERR_INVALID_PARMAMETER

Message Sent
"SBabcd" where **a**, **b**, **c**, and **d** are blocks of 4 streams each on the CT bus.

Response
None

Comments
This command configures the selected streams for the selected speed(s). Only the lower 16 streams are configurable on the CT bus.

CTBUS_CMD_QUERY_BOARD_CLOCK

command: CTBUS_CMD_QUERY_BOARD_CLOCK
device number: the device handle for the XDS board to receive the command
parameters:

&ctbus_query_h100_board_clock_parms (H.100/110 Boards)

struct ctbus_query_h100_board_clock_parms {

int size;	specifies the size of the struct used
int clock_type;	indicates the MVIP standard clocking used on the board
int clock_source;	specifies where the clock reference originates
int network;	the device source for the clock signals (if source == network)
int h100_clock_mode;	specifies the board's control of the H100 clocks
int auto_fall_back;	specifies whether the board is to automatically switch to the fall back mode and become a slave to alternate MC1 clock
int fall_back_occurred;	specifies whether the board has detected the primary master clock signal has become unreliable and fallen back to a secondary source
int h100_a_clock_status;	reports quality/status of the 'A' clock master signal
int h100_b_clock_status;	reports quality/status of the 'B' clock master signal
int netref_1_clock_status;	reports quality/status of the NETREF_1 clock secondary signal
int netref_2_clock_status;	reports quality/status of the NETREF_2 clock secondary signal

Applicable Boards
All XDS MVIP boards

Purpose
This command returns the clock modes.

Returns
CTBUS_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the current clock mode of the specified board. If config_8kref_clock and/or config_sec8k_clock are called before this function, this function will return "SUCCESS" and do nothing.

CTBUS_CMD_QUERY_BOARD_INFO

command: CTBUS_CMD_QUERY_BOARD_INFO
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_board_info_parms

struct ctbus_query_board_info_parms {

int size;	specifies the size of the struct used
int description[80];	receives the device driver description
int revision[16];	receives the revision level of device driver
int date[12];	release date of the device driver
int vendor[80];	receives the name of the vendor of the device driver
int serial_number[80];	receives the serial number of a specified board
int board_id;	receives the vendor-specific identity number
int base_port_address	receives the physical I/O address of board

}

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the board.

Returns
CTBUS_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the selected hardware. The serial_number field will always be “N/A”, no XDS boards have electronically embedded serial numbers. The date will always be 0000/00/00, again, no XDS boards have embedded dates. The base_port_address will always be 0xFFFFF, because of limitations of reading the hardware.

CTBUS_CMD_QUERY_DRIVER_INFO

command: CTBUS_CMD_QUERY_DRIVER_INFO
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_driver_info_parms

```
struct ctbus_query_driver_info_parms {
int size;                   specifies the size of the struct used
int description[80];        receives the device driver description
int revision[16];           receives the revision level of device driver
int date[12];               release date of the device driver
int vendor[80];             receives the name of the vendor of the device driver
}
```

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the driver.

Returns
CTBUS_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the device driver. The date will always be 0000/00/00, no XDS boards have electronically embedded dates.

CTBUS_CMD_QUERY_LOCAL_STREAM

command: CTBUS_CMD_QUERY_LOCAL_STREAM
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_local_stream_parms

```
struct ctbus_query_local_stream_parms {
int size;              specifies the size of the struct used
int local_stream;      the selected stream on local bus
int device_id;         device type on stream and timeslot selected
int parameter_id;      data item for configuration information obtained
int *buffer;           timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command is not supported by XDS boards.

Returns
CTBUS_SUCCESS
CTBUS_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will always return **CTBUS_ERR_NOT_CONFIGURABLE**.

CTBUS_CMD_QUERY_LOCAL_TIMESLOT

command: CTBUS_CMD_QUERY_LOCAL_TIMESLOT
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_local_timeslot_parms

```
struct ctbus_query_local_timeslot_parms {
int size;              specifies the size of the struct used
int local_stream;      the selected stream on local bus
int local_timeslot;    the selected timeslot on local bus
int device_id;         device type on stream and timeslot selected
int parameter_id;      data item for configuration information obtained
int *buffer;           timeslot-specific information from driver
}
```

Applicable Boards
No XDS boards

Purpose
This command is not supported by XDS boards.

Returns
CTBUS_SUCCESS
CTBUS_ERR_NOT_CONFIGURABLE

Message Sent
None

Response
None

Comments
This command is not compatible with XDS boards, and will always return **CTBUS_ERR_NOT_CONFIGURABLE**.

CTBUS_CMD_QUERY_OUTPUT

command: CTBUS_CMD_QUERY_OUTPUT
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_output_parms

```
struct ctbus_query_output_parms {
int size;                       specifies the size of the struct used
CTBUS_OUTDESC *output;          specifies the switch block outputs
};
```

Applicable Boards
All XDS boards.

Purpose
This command retrieves output information on a terminus.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_STREAM
CTBUS_ERR_INVALID_TIMESLOT
CTBUS_ERR_INVALID_MODE
CTBUS_ERR_INVALID_PARAMETER

Message Sent
None

Response
None

Comments
For all the XDS MVIP boards, this command interrogates tables to obtain the information. For MVIP streams, a single table is kept for all boards. For local streams including conferences and the MC1 bus, the driver checks the relevant table to return information on whether a timeslot is active or not, and what timeslot is the input or pattern is being output.

CTBUS_CMD_QUERY_STREAM_SPEED

command: CTBUS_CMD_QUERY_STREAM_SPEED
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_stream_speed_parms

```
struct ctbus_query_stream_speed_parms  {
int size;                    specifies the size of the struct used
int speed;                   specifies the speed of the specified stream
int *stream;                 specifies the stream(s) selected for query
};
```

Applicable Boards
XDS H.100 boards

Purpose
This command retrieves the speed of a specific stream.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_SPEED
CTBUS_ERR_INVALID_PARM

Message Sent
None

Response
None

Comments
This command reads dual-ported RAM for query information. Because of hardware limitations, streams are configured in blocks of four each (0-3, 4-7, 8-11, 12-15). So, this function will return the stream speed of each block, not an actual stream.

Example
When querying speed for stream 0, it will specify the speed for the first block (0-3). In addition, the MVIP95 specification limits the “speed” parameter to only one value, so when querying blocks that may have different speeds, this function may be called several times.

CTBUS_CMD_QUERY_SWITCH_CAPS

command: CTBUS_CMD_QUERY_SWITCH_CAPS
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_query_switch_caps_parms

struct ctbus_query_switch_caps_parms {

int size;	specifies the size of the struct used
int dvr_revision;	receives the revision level of the device driver (multiplied by 100)
int domain;	receives the domain of the switch block
int routing;	receives switch block's half duplex routing capabilities
int blocking;	receives switch block's possible blocking
int sw_standard;	the MVIP software standard being used
int sw_std_revision;	the revision of the MVIP software standard being used
int hw_standard;	the MVIP standard being used
int hw_std_revision;	the revision of the driver being used (multiplied by 100)
CTBUS_LOCAL_DEVICE_DESC *local_devs;	a pointer receiving the number of timeslots and device type of each local stream

}

Applicable Boards
All XDS MVIP boards

Purpose
This command returns information about the switch and its capabilities.

Returns
CTBUS_SUCCESS

Message Sent
None

Response
None

Comments
This command causes information to be returned in the structure that tells the application about the switching capabilities of the board. Note that the information is hard coded into the driver and is not returned by the board.

CTBUS_CMD_RESET_SWITCH

command: CTBUS_CMD_RESET_SWITCH
device number: the device handle for the XDS board to receive the command
parameters: none

Applicable Boards
All XDS boards

Purpose
This function can be used to put a board in a known, initialized state. All ports are released, all connections are broken, and all resources are freed. Outputs to the MVIP bus are disabled. This command does not change the clock mode of the board.

Returns
CTBUS_SUCCESS

Message Sent
"RA"

Response
All XDS boards respond with a message of type 2 subtype 1. The Switch Matrix Board makes no response.

Comments
This function should be used for all XDS boards when starting an application to put the boards in a known state. All connections are dropped and all resources are freed. The clock mode of the board is not altered by this command.

CTBUS_CMD_SAMPLE_INPUT

command: CTBUS_CMD_SAMPLE_INPUT
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_sample_input_parms

```
struct ctbus_sample_input_parms {
int size;                   specifies the size of the struct used
CTBUS_INDESC *input;        specifies the switch block inputs
};
```

Applicable Boards
All XDS Legacy/ISA boards, except the Switch Matrix board.

Purpose
This command retrieves the currently asserted byte on a switch block input.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_TIMESLOT
CTBUS_ERR_INVALID_STREAM
CTBUS_ERR_INVALID_PARAMETER

Message Sent
"QIsstt"

Response
None

Comments
This command causes the board to read the data memory of the FMIC chip to find the value asserted. In the case of the Multi-Chassis board, the board uses FMIC 2 to read the information if the stream is less than 0x13. Streams 0x10-0x13 are the local streams used by FMIC 2 to connect to the conference chips. If the stream number is greater than or equal to 0x14 the board will look for a connection on the MC1 bus for that stream and timeslot. If there is no such connection, then a value of 0xFF will be returned as the sample value. The Switch Matrix Board does not have an FMIC and does not support this command.

CTBUS_CMD_SET_OUTPUT

command: CTBUS_CMD_SET_OUTPUT
device number: the device handle for the XDS board to receive the command
parameters: &ctbus_set_output_parms

```
struct ctbus_set_output_parms {
int size;                       specifies the size of the struct used
CTBUS_OUTDESC *output;          specifies the switch block outputs
} ;
```

Applicable Boards
All XDS boards.

Purpose
This command is used to make and break connections, to disable a switch block output, or optionally, to continuously output a fixed pattern on a switch block output.

Returns
CTBUS_SUCCESS
CTBUS_ERR_INVALID_TIMESLOT
CTBUS_ERR_INVALID_STREAM
CTBUS_ERR_INVALID_PARAMETER

Message Sent
"MOssttiiiimpp" for Line boards and BRI boards
"SOssttiiiimpp" for the Multi-chassis board
where **sstt** is the output stream and timeslot, **iiii** is the input stream and timeslot, **m** is the mode and **pp** is the pattern value
"CLxxxyyy" for the Switch Matrix board in the connect mode
"CDxxx" for the Switch Matrix board in the disable mode where **xxx** is the output stream and timeslot and **yyy** is the input stream and timeslot

Response
None

Comments

The **CTBUS_CMD_SET_OUTPUT** command can be used to create connections using any of the switch blocks on the MC1 Multi-Chassis board. Streams 0x00-0x0F are the MVIP streams. Streams 0x10-0x13 are the local streams used to connect to the conferencing hardware. Streams 0x14-0x2B are the MC1 streams. Note, that to conference, additional commands must be issued to the board. A maximum of four streams may be used for transmitting to the MC1 bus. The messages to the board reflect this in that only streams numbered 0x14-0x17 are used. The library makes a translation from the range 0x14-0x2B to this range.

For the XDS Line boards, the **CTBUS_CMD_SET_OUTPUT** command controls the FMIC. It does not control either the seize function or the CODEC function of each port. To create a connection, an **XDS_MVIP_CONNECT** command must also be issued. The order of these commands is not important to the functioning of the board. To release a port, the **XDS_RLS** command must be used.

As the Switch Matrix board does not use an FMIC as the switch block, the actions of a **CTBUS_CMD_SET_OUTPUT** are approximated with the listen and disconnect messages to the board. There is no pattern capability on the Switch Matrix board. The DLL translates the streams in the **CTBUS_CMD_SET_OUTPUT** command to the appropriate values for the CL and CD commands used by the board. MVIP streams 0x0-0xF will map to streams 8-F on the board depending on the parameters sent to the **XDS_MX_SET_DIRECTION** command. MVIP streams 0x10-0x17 become 0-7 on the board. Streams 0-6 refer to the APIB connectors. Stream 7 is the PEB connector. Stream 6 may also be used to connect to the on-board DSPs.

MVIP-Related Error Codes

This page was intentionally left blank.

MVIP-90 Error Codes

General Errors		
SUCCESS	**0**	driver successfully completed command
MVIP_INVALID_COMMAND	**200**	command code is not supported
MVIP_DLL_INVALID_DEVICE	**201**	switch number passed to device driver DLL is out of range OS/2 specific
MVIP_DEVICE_ERROR	**202**	an error was returned by a device driver called by this device driver
MVIP_NO_RESOURCE	**203**	an internal device driver resource has been exhausted
Parameter Errors		
MVIP_INVALID_STREAM	**210**	stream number parameter is out of range
MVIP_INVALID_TIMESLOT	**211**	timeslot parameter is out of range
MVIP_MISSING_PARAMETER	**212**	not enough parameters to perform command
MVIP_INVALID_CLOCK_PARM	**213**	invalid clock configuration parameter(s)
MVIP_INVALID_MODE	**216**	invalid **SET_OUTPUT** or **QUERY_OUTPUT** mode
MVIP_INVALID_MINOR_SWITCH	**217**	invalid switch component in **dump_switch**
MVIP_INVALID_PARAMETER	**218**	other invalid parameter
Switch Errors		
MVIP_NO_PATH	**220**	connection cannot be made due to blocking or other switch limitation
MVIP_SWITCH_VERIFY_ERROR	**221**	verification of switch operation failed
MVIP_INTERNAL_CONFLICT	**222**	more than one switch component is in conflict
MVIP_CONNECTION_NOT_SUPPORTED	**223**	switch block does not support connection

MVIP-95 Error Codes

General Errors		
MVIP95_SUCCESS	**0**	driver successfully completed command
MVIP95_ERR_INVALID_COMMAND	**200**	command code is not supported
MVIP95_ERR_DLL_INVALID_DEVICE	**201**	DLL could not find specified device
MVIP95_ERR_DEVICE_ERROR	**202**	an error was returned by a device driver called by this device driver
MVIP95_ERR_NO_RESOURCES	**204**	an internal device driver resource has been exhausted
Parameter Errors		
MVIP95_ERR_INVALID_STREAM	**210**	stream number parameter is out of range
MVIP95_ERR_INVALID_TIMESLOT	**211**	timeslot parameter is out of range
MVIP95_ERR_MISSING_PARAMETER	**212**	not enough parameters to perform command
MVIP95_ERR_INVALID_CLOCK_PARM	**213**	invalid clock configuration parameter(s)
MVIP95_ERR_INVALID_SPEED	**214**	speed parameter is out of range
MVIP95_ERR_NOT_CONFIGURABLE	**215**	device does not support configuration of parameters/values requested
MVIP95_ERR_INVALID_MODE	**216**	invalid **SET_OUTPUT** or **QUERY_OUTPUT** mode
MVIP95_ERR_INVALID_MINOR_SWITCH	**217**	invalid switch component in **dump_switch**
MVIP95_ERR_INVALID_PARAMETER	**218**	other invalid parameter
MVIP95_ERR_UNSUPPORTED_MODE	**224**	mode not supported by device driver or the hardware specified
Switch Errors		
MVIP95_ERR_NO_PATH	**220**	connection cannot be made due to blocking or other switch limitation
MVIP95_ERR_SWITCH_ VERIFY_ERROR	**221**	verification of switch operation failed
MVIP95_ERR_INTERNAL_CONFLICT	**222**	more than one switch component is in conflict
MVIP95_ERR_CONNECTION_ NOT_SUPPORTED	**223**	switch block does not support connection

CT-BUS Error Codes

General Errors		
CTBUS_SUCCESS	**0**	driver successfully completed command
CTBUS_ERR_INVALID_COMMAND	**200**	command code is not supported
CTBUS_ERR_DLL_INVALID_DEVICE	**201**	DLL could not find specified device
CTBUS_ERR_DEVICE_ERROR	**202**	an error was returned by a device driver called by this device driver
CTBUS_ERR_NO_RESOURCES	**204**	an internal device driver resource has been exhausted
Parameter Errors		
CTBUS_ERR_INVALID_STREAM	**210**	stream number parameter is out of range
CTBUS_ERR_INVALID_TIMESLOT	**211**	timeslot parameter is out of range
CTBUS_ERR_MISSING_PARAMETER	**212**	not enough parameters to perform command
CTBUS_ERR_INVALID_CLOCK_PARM	**213**	invalid clock configuration parameter(s)
CTBUS_ERR_INVALID_SPEED	**214**	speed parameter is out of range
CTBUS_ERR_NOT_CONFIGURABLE	**215**	device does not support configuration of parameters/values requested
CTBUS_ERR_INVALID_MODE	**216**	invalid **SET_OUTPUT** or **QUERY_OUTPUT** mode
CTBUS_ERR_INVALID_MINOR_SWITCH	**217**	invalid switch component in **dump_switch**
CTBUS_ERR_INVALID_PARAMETER	**218**	other invalid parameter
CTBUS_ERR_UNSUPPORTED_MODE	**224**	mode not supported by device driver or the hardware specified
Switch Errors		
CTBUS_ERR_NO_PATH	**220**	connection cannot be made due to blocking or other switch limitation
CTBUS_ERR_SWITCH_ VERIFY_ERROR	**221**	verification of switch operation failed
CTBUS_ERR_INTERNAL_CONFLICT	**222**	more than one switch component is in conflict
CTBUS_ERR_CONNECTION_ NOT_SUPPORTED	**223**	switch block does not support connection

XDS Error Codes

NO_BRD	**001**	board not present at given number
NO_RSPND	**002**	board not responding
ILL_PORT	**003**	invalid port number
ILL_SLOT	**004**	invalid timeslot
ILL_ARG	**005**	invalid parameter for XDS command
ILL_HAND	**006**	invalid conference handle
ILL_ATTEN	**007**	invalid attenuation value
ILL_THRES	**008**	invalid threshold value
WRONG_QUERY	**009**	query problem
WRONG_BOARD	**010**	illegal board selected
NO_UPDATE	**011**	code returned from functions that don't return an error
ILL_CCA	**012**	illegal conference control address
BRD_ERROR	**-1**	board operation problem