# XDS Windows 2000/XP
# H.110 Driver
# Reference Manual

**Driver Version 1.5**
**November 2004**

**American Tel-A-Systems, Inc.**

**258M011F ©**

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**

# XDS Windows 2000/XP H.110 Driver Reference Manual

**Author: Brian D. Riek**


**American Tel-A-System, Inc.**
**608-838-4194**
**4800 Curtin Drive, McFarland, WI 53558, USA**
http://www.amtelco.com/
**258M011F**

# Driver Package Software Installation And Removal

This page was intentionally left blank.

**Driver Package Contents** -

The XDS Windows 2000/XP H.110 Driver package comes in the form of a CD-ROM disc (Amtelco P/N 258CD004) or self-extracting executable - if downloaded. This disc/image contains the device driver along with an .inf file (which is used for the initial driver installation), the WISE installer installation program for the driver application suite and the source code. If downloaded, the user will need to install the driver package first, it will then copy the PCI low-level driver (xds_2000_110.sys) into the \Program Files\Amtelco\H110\ directory along with the .inf file.

# 1.0 Hardware Installation

If the chassis does not support PCIXCAP or M66EN (which are defined in the Hot Swap specification PICMG 2.1 R2.0), then jumper JW4 on the XDS H.110 board will need to be installed. A good indication of this would be if, once the board is plugged in, that the blue hot-swap LED remains on.

Each XDS H.110 board uses 8K of memory and comes in the cPCI form factor. The resources for each PCI device in the system can be viewed in the system BIOS at boot-up.

You will need to be sure that there is a PCI interrupt available for the cPCI board(s).

As with all device drivers in most operating systems, the user must have administrator privileges in order to install/remove a device driver.

You will need to power down the system that the board(s) will be installed in. Make sure to save any work that you may have been doing. Follow the board's hardware manual precisely for the board installation portion. When this step is completed, power the system back on.

# 2.0 Driver Installation

After Windows is finished starting up, the *Found New Hardware* dialog box will appear (Figure 1.0).


**Figure 1.0**

Now another window (Figure 1.1), the *Found New Hardware Wizard*, will appear over the first one. Click on the **Next >** button.


**Figure 1.1**

The next window is the *Install Hardware Device Drivers* window (Figure 1.2). Select the **Search for a suitable driver for my device (recommended)** option and click on the **Next >** button.


**Figure 1.2**

At this point of the installation, insert the driver disc into the CD-ROM drive of the system. Now you will need to locate the driver files (Figure 1.3). Select the **CD-ROM drives** option and click on the **Next >** button.


**Figure 1.3**

The next window should look like Figure 1.4 when “xds_2000_110.inf” is found on the disc.  Simply click on the **Next >** button now.


**Figure 1.4**

The last window to appear during the install of a new device is like the one pictured in Figure 1.5. Now, click on the **Finish** button.

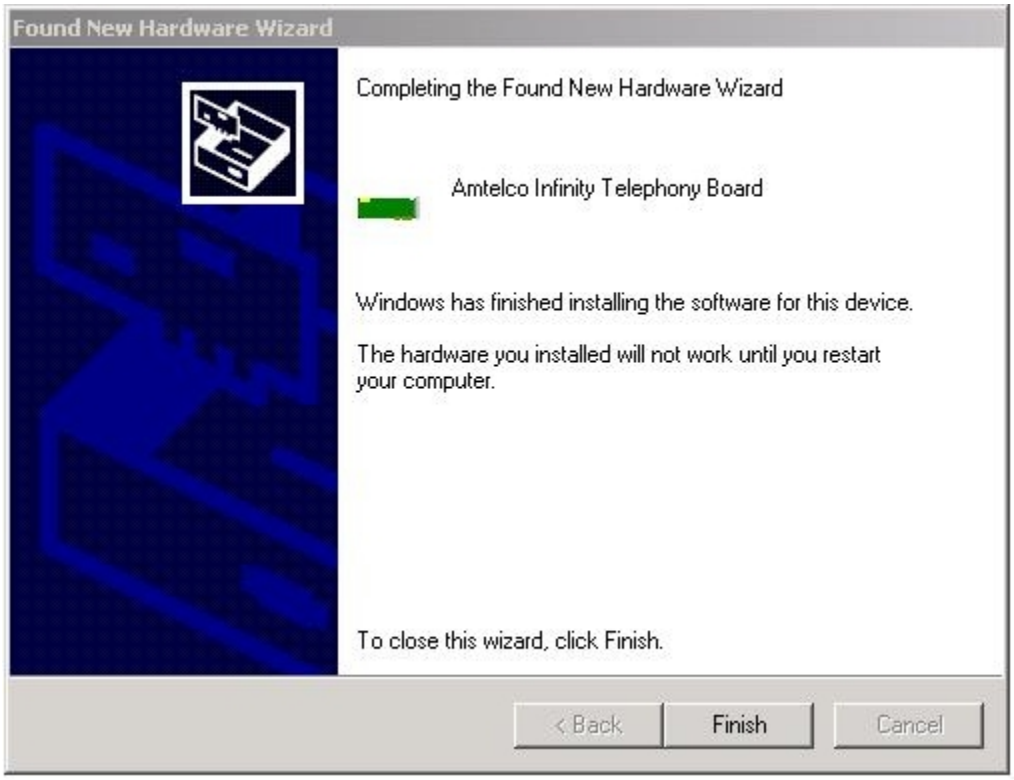


**Figure 1.5**

That concludes the installation process of the device driver. Each additional board will go from step 1 (Figure 1.0) immediately to the last step (Figure 1.5) and require no further intervention.

When Windows 2000/XP starts up, it will assign the memory offset, IRQ, and I/O port dynamically for each PCI board. These settings may be viewed in the **Device Manager**. Once the **Device Manager** is open, you will notice that the each XDS H.110 board will appear under the Computer Telephony Device class (Figure 2.0).

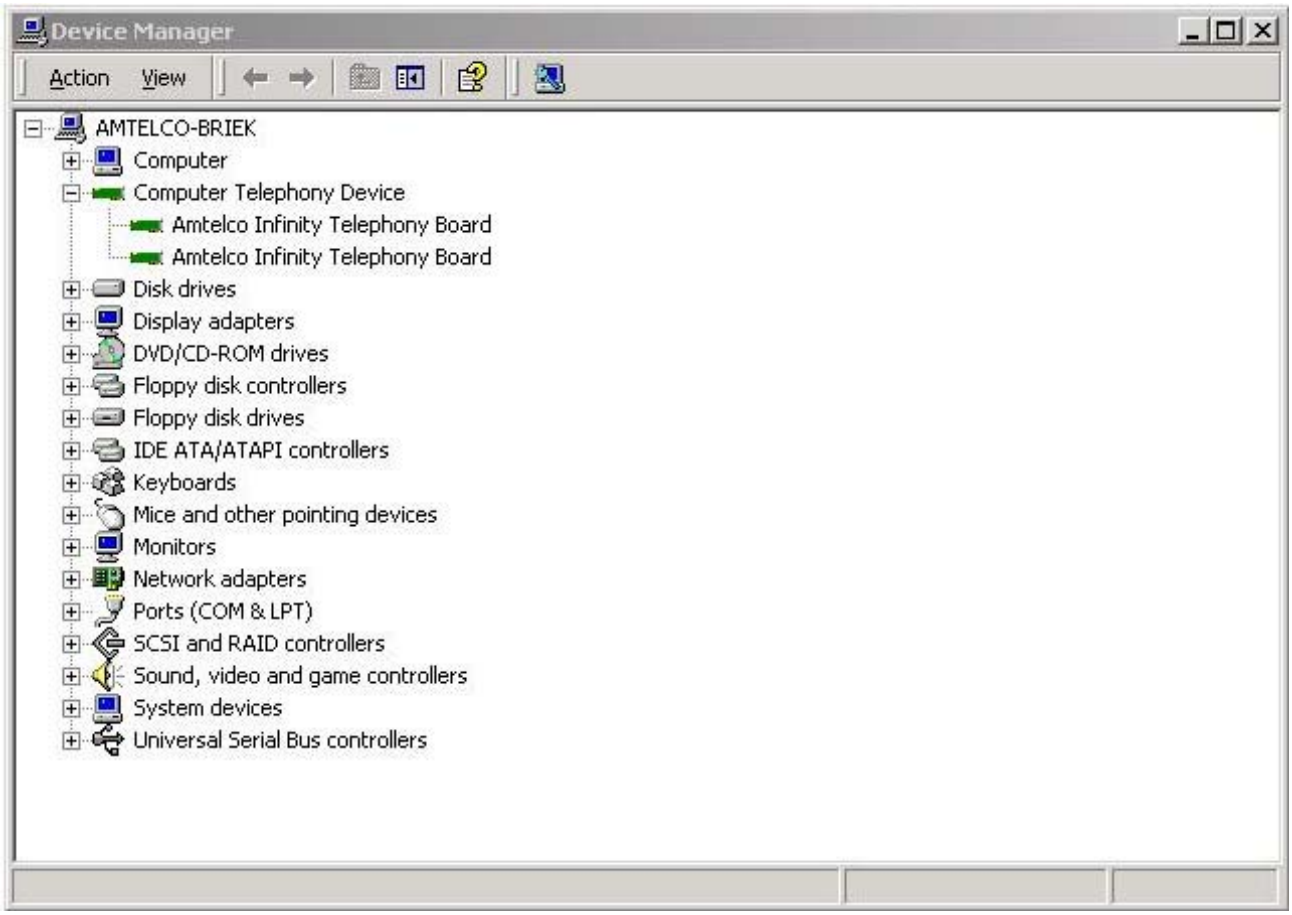


**Figure 2.0**

The parameter values will be saved in the Windows Registry and should never be modified or removed directly by the user. These parameters are saved at:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\XDS_2000_110

If problems occur with the driver, they will be identified in the **Event Viewer** with the *Event Source* "XDS_2000_110" and an Event Code.

# 3.0 Driver Package Installation

You are now ready to install the driver package (application suite and source code). This procedure will use a installation wizard setup created with WISE InstallMaster (version 8.0), which will guide you step by step with instructions.

Click on the **Start** menu button, and select the **Run…** command. Click on **Browse…** and locate your CD-ROM drive. When you locate the CD-ROM, highlight the **setup.exe** file, and click the **Open** button. This will begin the setup wizard.

When finished, the setup will have created a start menu item for your applications.

# 4.0 Driver(s) and Driver Package Removal

To ensure the proper removal of the XDS driver package, both drivers (if both were installed), and any other components; please follow the following steps in order:

1) Close all XDS-related programs and project workspaces, if open. Save any work necessary to your development.

**2a**) First, remove the device driver from the system. You will need to run the **Add/Remove Hardware** utility in the **Control Panel** (Figure 3.0) and click **Next >**.


**Figure 3.0**

2**b**) Next, select the **Uninstall/Unplug a device** option and click on **Next >** (as in Figure 3.1).

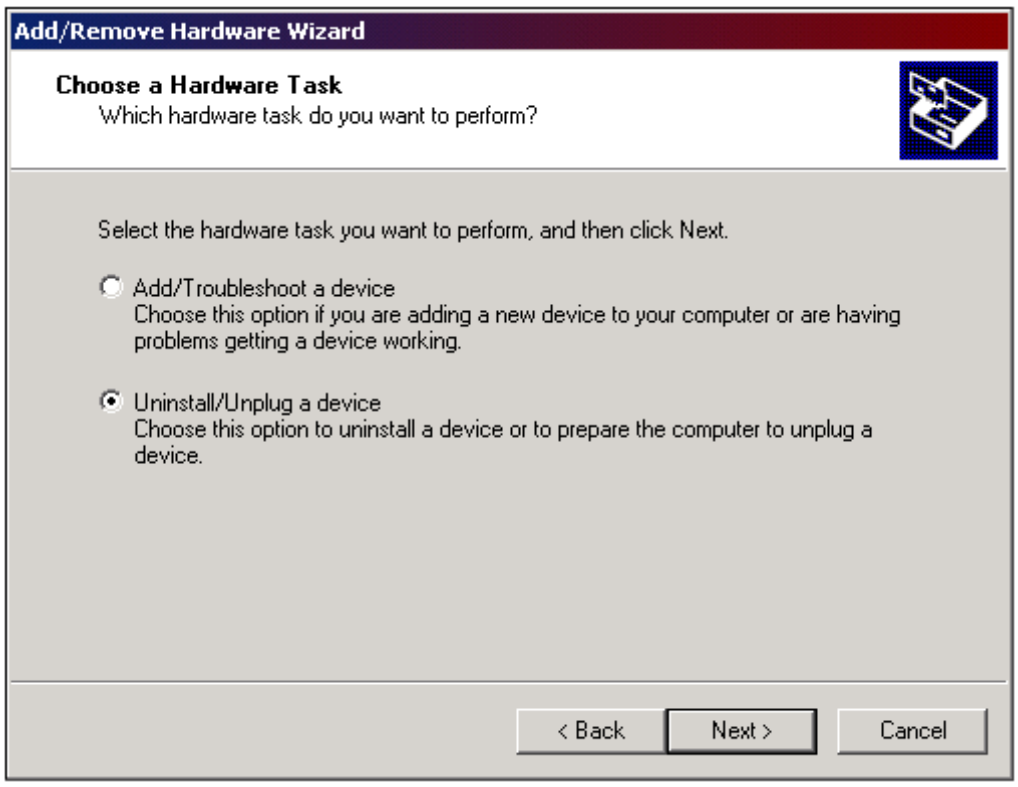


**Figure 3.1**

**2c**) Then, select **Uninstall a device** and click on **Next >** in the next window (like the one in Figure 3.2).

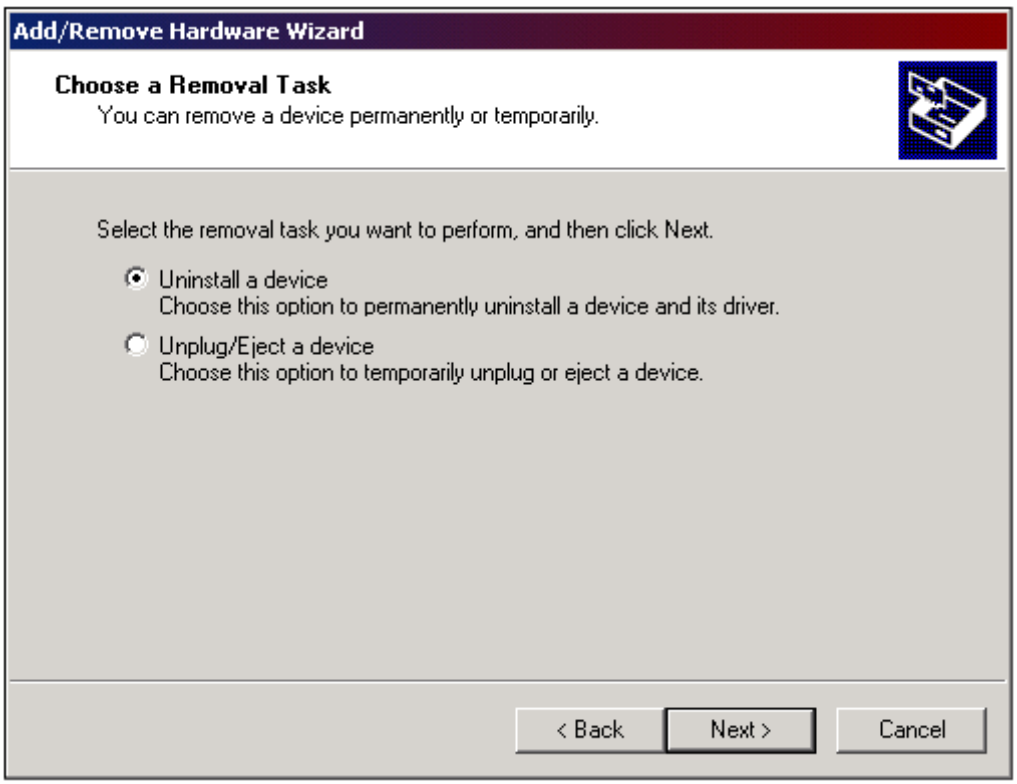


**Figure 3.2**

**2d)** You will now need to select the **Amtelco H.110 Infinity Telephony Board(s)** from the device list (as pictured in Figure 3.3) and click **Next >**.

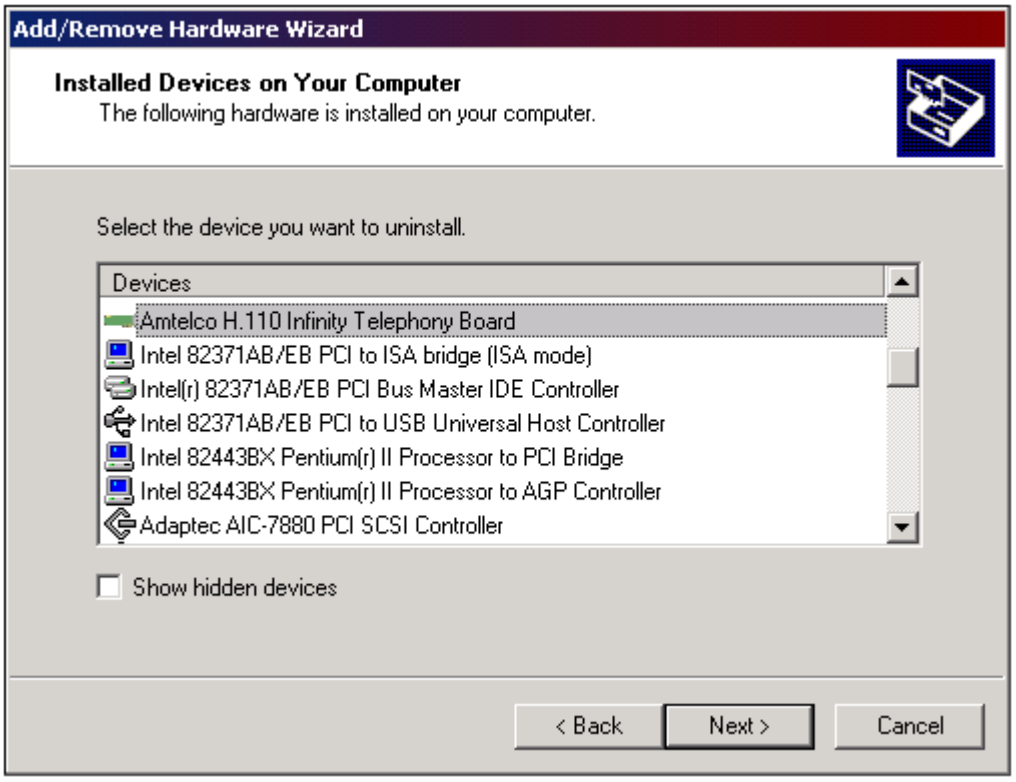


**Figure 3.3**

**2e**) Select **Yes, I want to uninstall this device** option from the following window (Figure 3.4) and click on **Next >**.

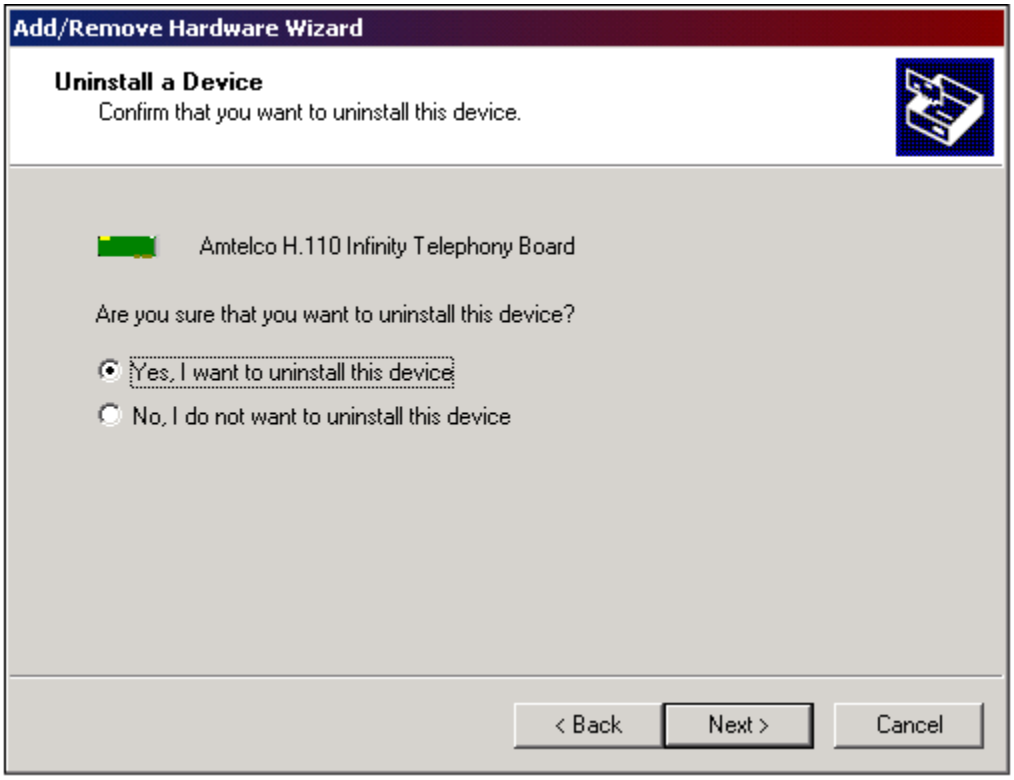


**Figure 3.4**

2**f**) Click on **Finish >** button in the last window (Figure 3.5).

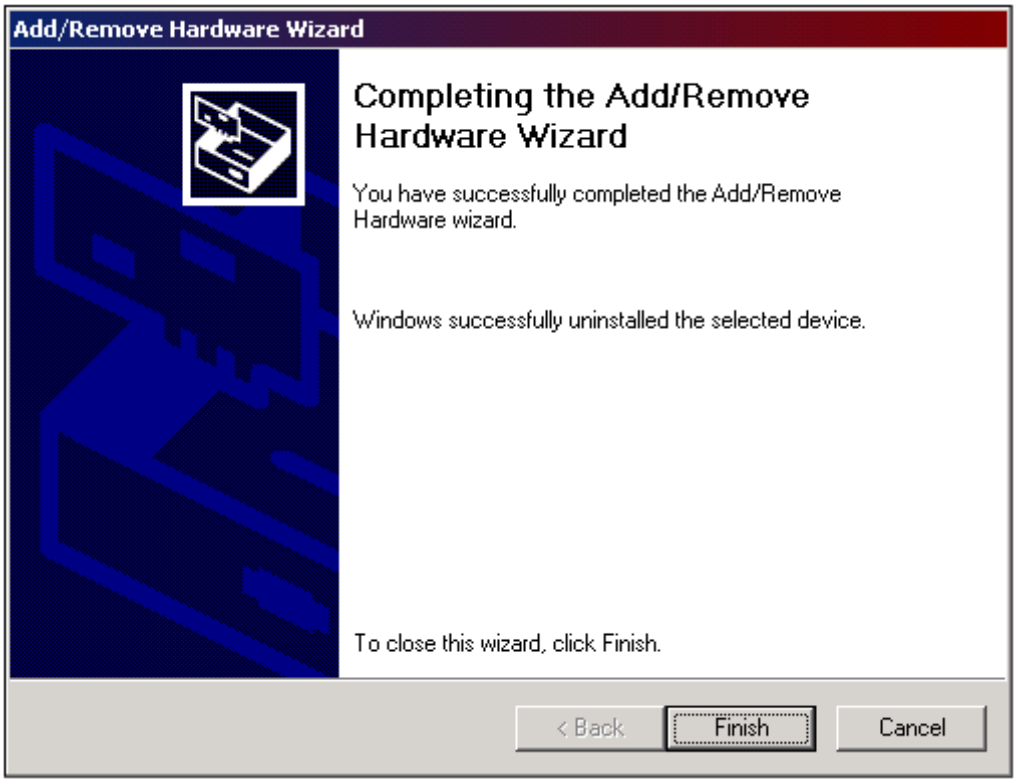


**Figure 3.5**

1) Now, you may un-install the application suite and source code from the system. You will need to run the **Add/Remove Programs** utility in the **Control Panel** in order to do this. The driver package will be listed as the "**XDS Win 2K/XP H.110 Driver Pkg"**.

2) Power off you system and remove any XDS hardware installed.

3) Finished!!!

# 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 defines (ie: "XDS_H110" and "XDS_2000_ARC") 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 2000 DDK were used to compile and build the low-level driver (xds_2000_110.sys).

# 2.0 Tests / Utilities

**NOTE: All message strings sent to any board, using any one of the provided utilities, must be in CAPITAL letters. In addition, only application should be opened at a time!**

**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 on 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. This demo program uses a "polling" scheme of receiving messages from a board, and is less efficient than one of the interrupt-driven demos, such as Sig_Util.

**Driver Command Line Test**

The test program **test_drv** is a simple program, written in 'C', that demonstrates how to make PCI 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, or just "**test_drv**" to display a list of XDS boards to use. The program first displays any messages that might be already on the board's queue(s). Then, the options: **s = send, r = receive,** 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. 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. 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.

A button, labeled “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. This is one of the more efficient of the XDS demo programs, and is recommended to model the users program around. It is a good example of how an application uses the XDS Native Library function set in conjunction with the drivers’ signaling events. Using interrupts is much more efficient than polling for messages, and the user should keep the design philosophy in mind when writing ones own application.

**Sig_Util2 (2 Board GUI Communication Utility w/Signaling)**

A Graphical User Interface, Sig_Util2, has been provided for multiple board communication with that includes signaling. There is a pull-down list used to select which two boards to transmit messages to. The user may choose any two boards at any time during run-time. Receive messages (from the board) for the first board “Board 1” are displayed in the *Board 1 receive messages* window, and receive messages (from the board) for the second board “Board 2” are displayed in the *Board 2 receive messages* window. Transmit messages (to the board) for the first board “Board 1” are entered in the *Board 1 transmit message* edit box, and transmit messages (to the board) for the second board “Board 2” are entered in the *Board 2 transmit message* edit box.

The **signal_test** program 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**

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 just "test_dll" to display a list of XDS boards to use. The program will first display any messages that might be already on the board's queue(s). Then, the options: **s = send, r = receive, l = send_layer3_msg,** 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.

To send a Layer 3 test message (intended for BRI boards only), type in '**l**' and then press the "Enter" key. 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 terminates.

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, or just "testmv90" to display a list of XDS boards to use. The **testmv90** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

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, or just "testmv95" to display a list of XDS boards to use. The **testmv95** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

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, or just "testctbus" to display a list of XDS boards to use. The **testctbus** program communicates with the DLL and displays the response of several high-level commands that are sent to the XDS board.

**XdsPciRes**
The XdsPciRes program is a command line utility that takes no parameters and simply displays each PCI board number, board ID code, PCI bus number, and PCI device/function (slot) number.

**XDS_BRI_Config**
When you start the program, the first window will show the board number, board ID, version string, and number of ports each XDS BRI boards in the system. If no XDS BRI board is found in the system, the program will exit.

You can choose the board number that you want to initialize from the combo box and click the "config" button to start a configuration window.

1. Choose the protocol layer for each port.
North American (NI-1/NI-2): Layer 2, Layer3, AT&T Custom, CACH_EKTS, DMS-100, or National ISDN.
EURO-ISDN: Layer 2, Layer 3, or Point-to-Point.

2. Choose the port type for each port.
For the S/T board: choose "TE" for terminal equipment, choose "NT" for network terminations, and choose UNDEFINED for not used ports. For the U-Interface board: choose "LT" for line termination ports and choose "NT" for network terminations.

3. Enter the Directory Number and SPID for each B Channel.
Each port has two (2) B channels. For the "NT" ports, you only need to enter the directory number. For the "TE" ports, you need to enter both Directory Number and SPID number.

4. There are three options, with check boxes. Auto TEI Assignment and TEI Check Response format for North American (NI-1 & NI-2) and Incoming Address Checking for Euro ISDN.

5. If you want the data to automatically be set each time the system is booted, you check the "save on board" check box.

6. If the number of ports is greater than 16 (ie: H.110 board), you should click the

"Port 10-1F" button to set the data for ports 0x10 to 0x1F.

7. After you have done all data entry, you should click "Ok" button. The program will get all data and send it to the board.

8. You can save all the initialization data into an ASCII file (on your PC) by clicking the "Save to File" button. This file will be saved in the local directory with the extension "cfg". Next time, you can retrieve the data from an existed file by clicking the button "Retrieve".

For more information about the XDS Basic Rate ISDN Board, please refer to the BRI technical manual for the appropriate board.

**MC-3 Fiber Ring Integrity Test**

The test program Mc3_Fiber_Test.exe is a test utility that tests ring integrity between two H.100 or H.110 MC3 boards. It is a two-part (side) process with three steps on each chassis, that needs some user-intervention. The program will first initialize each board by setting up the encoding mode, clock mode, and ring mode for each.

The user will then follow these steps in order:
1) Designate which "side" chassis will be the receive and which will be the transmit.
2) On the transmit side - type in "**mc3_fiber_test n"**, where **n** is the number of an installed XDS board, in a command prompt window.

3) Now select the 'T' (transmit) operating mode. This will send a pattern of "55" to the receive chassis.

4) On the receive side - type in "**mc3_fiber_test n"**, where **n** is the number of an installed XDS board, in a command prompt window.

5) Now select the 'R' (receive) operating mode. This will display the pattern received to the user. It will then instruct the user to go back to the transmit chassis and send the next pattern.
6) On the transmit side enter a 'Y' if the correct pattern, "55", was received by the receive chassis. Now the transmit side will send a pattern of "FF" to the receive

chassis.

7) On the receive chassis, hit the ‘Enter’ key. This will display the pattern received to the user. The pattern here should now be “FF”. It will then instruct the user to go back to the transmit chassis and send the next pattern.

8) On the transmit side enter a ‘Y’ if the correct pattern, “FF”, was received by the receive chassis. Now the transmit side will send a pattern of “AA” to the receive chassis.

9) On the receive chassis, hit the ‘Enter’ key. This will display the pattern received to the user. The pattern here should now be “AA”.

If any of the patterns received in any one of the receive steps was not what it was suppose to be, then re-check your fiber connections and try this program again. If it does not work after that, then report this problem to an Amtelco XDS Field Engineer or Customer Service representative.

# 3.0 Hot-plugging Hardware

All of the XDS H.110 boards have the ability to be removed while the system is up and running and then replaced. The user must keep in mind that boards can only be replaced with the same type of board in the same physical slot as the one removed. So, if the user removes an XDS H.110 MC3 board from slot 3 in the system, they **must** replace it with another XDS H.110 MC3 board in that same physical slot. If the user has a 2 piece board set, like the XDS H.110 BRI S/T board, they would remove the front board first, then the rear I/O board second. When replacing, the rear I/O board would be inserted first and the front board second.

An example application, **tstchs,** was written to demonstrate to the user who to properly call each IOCTL function. The syntax is "**tstchs n"**, where **n** is the number of an installed XDS board, or just "tstchs" to display a list of XDS boards to use. The user options are displayed during run-time. To remove an XDS H.110 board using 'tstchs', the user will use the 'd' option from the menu. Then the user will remove the board and replace it with the new board (in the same H.110 chassis slot). Lastly, the user will use the 'i' option from the menu to insert the board.

# 4.0 Downloader

Most of the XDS boards are equipped with flash memory, which contains the board program. Refer to the board reference manual to check for this feature. 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 1

will flash the firmware file, 258H001.hex, to the control processor onto board 1.

# 5.0 DLL Descriptions

**XdsLib110 DLL**
An "XDS" DLL (XdsLib110.dll) has been provided to access XDS H.110 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.

# 6.0 Source Code And Directory Structure

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

Binary file directory -

| | |
|---|---|
| \H110\bin\intel | - executables, DLLs, driver, and downloader |

Source code directories -

| | |
|---|---|
| \H110\source\Dlls\XdsLib110 | - xdslib110.dll (XDS H.110 native functions) |
| \H110\source\Dlls\XdsMv90 | - xdsmv90.dll (MVIP-90 functions) |
| \H110\source\Dlls\XdsMv95 | - xdsmv95.dll (XDS functions) |
| \H110\source\Dlls\XdsCtBus | - xdsctbus.dll (XDS functions) |
| \H110\source\Downloader | - wn386dlc downloader program |
| \H110source\Mc3_Fiber_Test | - MC3 fiber ring test source code |
| \H110\source\Driver | - xds_2000_110.sys low-level driver |
| \H110\source\Include | - include (header) files |
| \H110\source\Lib | - library files for Intel x86 processors |
| \H110\source\Wise | - Wise installation project file |
| \H110\source\Shared | - shared source code directory |
| \H110\source\Sig_Util | - Sig_Util application source code |
| \H110\source\Sig_Util2 | - Sig_Util2 application source code |
| \H110\source\Signal_Test | - Signal_Test application source code |
| \H110\source\Test_dll | - xdslibmv.dll test |
| \H110\source\TestMv90 | - xdsmv90.dll test |
| \H110\source\TestMv95 | - xdsmv95.dll test |
| \H110\source\TestCtBus | - xdsctbus.dll test |
| \H110\source\Tstchs | - tstchs application source code |
| \H110\source\Test_drv | - test_drv.exe driver test |
| \H110\source\XdsUtil | - xdsutil (utility) application |
| \H110\source\XDS_BRI_Config | - xds_bri_config (utility) application |
| \H110\source\Station_Config | - station_config (utility) application |
| \H110\source\XdsPciRes | - xdspcires (utility) application |

# XDS Windows 2000/XP
# H.110 Driver
# IOCTL Description

This page was intentionally left blank.

# Overview

The XDS Windows 2000/XP H.110 Driver is designed to provide an interface between XDS boards and applications running under Windows 2000 and Windows XP. 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 CAPITAL 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 DevIoControl supports the following 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_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 | - de-activate board and remove function |
| XDS_RESUME | - re-activate board and insert function |
| **XDS_GET_BUS_DEVICE_NUM** | - obtain the PCI bus and slot number for a given XDS PCI board |
| XDS_QUEUE_USER_MSG | - place an ASCII message on the receive message queue |
| XDS_GET_BOARD_INFO | - get board ID, version, and number of ports |
| READ_PLX_INT | - read the PLX (PCI) interrupt register status |

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. 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 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.

# cPCI Application Interface

Applications can interface directly to the driver by using the Windows 2000/XP system calls GetDeviceViaInterface, CloseHandle, and DevIoControl. 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 signaling mechanism.

GetDeviceViaInterface
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. All event queues will be initialized whenever a connection with the XDS driver is opened.

GetDeviceViaInterface((LPGUID)&XDS_IO_GUID, 0);

The XDS_IO_GUID symbol refers to the Amtelco XDS Computer Telephony Class (16F4A638-1B29-435B-85A1-0077D14CD8B2).

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

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 |
| **&data_length,** | 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
&data_length,               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
&data_length,               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
**&data_length,** 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 |
| **&data_length,** | 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_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
&data_length,               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 |
| **&data_length,** | 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_GET_BUS_DEVICE_NUM

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | device handle |
| **(DWORD) XDS_GET_BUS_DEVICE_NUM,** | board ID command |
| **pData,** | pointer to input structure |
| **sizeof(XDS_BOARD_INFO),** | length of input structure |
| **pData,** | pointer to output structure |
| **sizeof(XDS_BOARD_INFO),** | length of output structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | |

XDS_BOARD_INFO info;

typedef struct{

| | |
|---|---|
| char version[4]; | the firmware version of a board |
| char id[4]; | the ID of a board |
| UCHAR num_ports; | number of ports |
| UCHAR board_number; | board number |
| ULONG pci_bus_number; | PCI bus number |
| ULONG pci_slot_number; | PCI slot number |

} XDS_BOARD_INFO, *PXDS_BOARD_INFO;

**Purpose**
This command is used to obtain the PCI bus and slot number of a specified board.

**Returns**
The function will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| XDS_ERR_GET_BUS_DEVICE_NUM | error obtaining the PCI slot and bus number |

**Comments**
This function is for PCI H.100 boards only.

# XDS_SLEEP

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | device handle |
| **(DWORD)XDS_SLEEP,** | hardware “sleep” remove board command |
| **pData,** | pointer to input structure |
| **sizeof(XDS_MSG),** | length of input structure |
| **NULL,** | pointer to output structure |
| **0,** | length of output structure |
| **&data_length,** | 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 stop a board’s interrupts and prepare it to be physically removed from a chassis.

**Returns**
The function will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| XDS_ERR_IOCTL_SLEEP | error removing board, board may not be functioning properly |

**Comments**
The board should not be actively being used at the time of this function call. Therefore, the user should exit all applications that may be communicating with the board.

In order for hot-plugging a board to work, the user must insert the same type of board in the same physical slot as the one that was removed. If the board has a rear I/O board, the user will remove the front board first, then the rear board. Then the user will replace the rear I/O board and insert the front one.

# XDS_RESUME

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | device handle |
| **(DWORD)XDS_RESUME,** | hardware “resume” insert board command |
| **pData,** | pointer to input structure |
| **sizeof(XDS_MSG),** | length of input structure |
| **NULL,** | pointer to output structure |
| **0,** | length of output structure |
| **&data_length,** | 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 add a board and prepare it to be used in place of an identical board that is in the same physical slot as the one removed.

**Returns**

The function will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| XDS_ERR_IOCTL_RESUME | error removing board, board may not be functioning properly |

**Comments**

In order for hot-plugging a board to work, the user must insert the same type of board in the same physical slot as the one that was removed. If the board has a rear I/O board, the user will remove the front board first, then the rear board. Then the user will replace the rear I/O board and insert the front one.

# XDS_QUEUE_USER_MSG

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | device handle |
| **(DWORD)XDS_QUEUE_USER_MSG,** | IOCTL low-level driver command |
| **&msg,** | pointer to message structure |
| **sizeof(XDS_MSG),** | length of message |
| **NULL,** | pointer to output structure |
| **0,** | length of output structure |
| **&data_length,** | 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 the board's receive message queue. 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**
Several library functions use this call when a port may be on hold and an "SBxx" message needs to be returned to the user in the message receive queue. This is also available to be used by the user.

# XDS_GET_BOARD_INFO

**BOOL DevIoControl(**
**hdriver,** device handle
**(DWORD) XDS_GET_BOARD_INFO,** board INFO command
**pData,** pointer to input structure
**sizeof(XDSID),** length of input structure
**pData,** pointer to output structure
**sizeof(XDSID),** length of output structure
**&data_length,** pointer to number of bytes returned
**NULL);**

XDSID info;

typedef struct xdsid {
unsigned char board_number; board number
char id[5]; board type (ID)
char version[5]; firmware version
int number_ports; number of ports
UCHAR pci_device_number; PCI Board device number
UCHAR pci_bus_number; PCI Board bus number
}XDSID, *PXDS_ID, xiID, *pXdsId;

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

**Returns**
The function will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_DATA_ERROR | board number used, not valid |

**Comments**
This function returns the board ID, version, and number of “ports” associated with a specified XDS board.

# READ_PLX_INT

**BOOL DevIoControl(**
**hdriver,** device handle
**(DWORD) READ_PLX_INT,** read PLX INT command
**pData,** pointer to input structure
**sizeof(XDSINTCSR),** length of input structure
**pData,** pointer to output structure
**sizeof(XDSINTCSR),** length of output structure
**&data_length,** pointer to number of bytes returned
**NULL);**

```
typedef struct xdsintcsr
{
        char plx_9030;          // a char to tell the app if it is a PLX 9030 or other
        char board_number;      // board number
        char interrupt_enable1;         // byte 1 of the interrupt register
        char interrupt_enable2;         // byte 2 of the interrupt register
        char interrupt_enable3;         // byte 3 of the interrupt register
        char interrupt_enable4;         // byte 4 of the interrupt register
}       XDSINTCSR, *PXDSINTCSR;
```

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

**Returns**
The function will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_DATA_ERROR | board number used, not valid |

**Comments**
This function returns the PLX interrupt register value for a specified XDS board.

# 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/XP Implementation**
The specific implementation for Windows NT, Windows 2000, and Windows XP 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/XP 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-90 Software
# Command Reference

This page was intentionally left blank.

# CONFIG_CLOCK

**command: CONFIG_CLOCK**
**device number:** the device handle for the XDS board to receive the command
**parameters: &clock_param**

| | |
|---|---|
| struct clock_param { | a structure specifying the clock mode |
| byte clock; | specifies the clock circuit operating mode<br>0x00 - clock reference comes from MVIP bus /FS<br>0x01 - clock reference comes from the MVIP bus SEC8K<br>0x02 - clock reference comes from on board oscillator<br>0x80 - clock reference comes from a network connection |
| byte sec8K; | specifies whether this device drives the MVIP bus SEC8K line<br>0x01 - SEC8K not driven<br>0x02 - SEC8K driven by on board oscillator<br>0x80 - SEC8K is driven by clock of network connection |
| int network; | specifies which network connection is the source of the 8KHz reference timing for either the master clock or the SEC8K line |
| } | |

**Applicable Boards**
The XDS Switch Matrix, DID, E&M, Ground Start, Loop Start, and Station Boards.

**Purpose**
This function is used to set the clock mode for a board.

**Returns**
**SUCCESS**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_CLOCK_PARM**
**MVIP_INVALID_PARAMETER**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**
**XDS_INVALID_BOARD**

**Message Sent**
“SCx” for the Switch Matrix and Line boards where x is the clock mode

**Response**
None

**Comments**
For line boards, the clock modes available correspond to the modes 0-2 for the clock parameters,

there is no network clock connection. The SEC8K output is not separately controllable. The clock modes for the Multi-Chassis board are much more complex as there are interactions with the MC1 bus. Therefore, the clocks for both the MVIP and MC1 busses are configured using the **MC1_CONFIG_CLOCK** command.

Note that the clock modes sent to the Switch Matrix board in the "**SC"** message do not match the mode in the **clock** parameter. The following codes are used in the message:

0 - no clock to or from the MVIP bus, the Matrix Board is the Master clock
1 - Matrix Board provides clock to the MVIP bus
2x - APIB highway x provides the master clock (network is clock master)
3 - MVIP bus is the master clock

# DUMP_SWITCH

**command: DUMP_SWITCH**
**device number:** the device handle for the XDS board to receive the command
**parameters: &dump_parms**

struct dump_parms { a structure to contain the return information

| | |
|---|---|
| int size; | size of the structure in bytes |
| int minorsw; | specifies the switch component to be read |
| int stream; | specifies the input and output stream numbers |
| int timeslot; | specifies the timeslot, typically, 0-31 |
| int cmhi; | receives the contents of the connection memory HI byte |
| int cmlo; | receives the contents of the connection memory LO byte |
| int data; | receives the contents of the data memory read |

}

**Applicable Boards**
XDS Multi-Chassis, DID, E&M, Ground Start, Loop Start, & Station boards. This function is not supported by the XDS Switch Matrix board.

**Purpose**
This command retrieves the contents of a switch component within an MVIP switch block. In general, this is hardware specific. This information is useful for diagnostic purposes.

**Returns**
**SUCCESS**
**MVIP_INVALID_STREAM**
**MVIP_INVALID_TIMESLOT**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**
**XDS_INVALID_BOARD**

**Message Sent**
"QDsstt" for XDS MVIP Line boards where **ss** is the stream and **tt** is the timeslot
"QDfsstt" for XDS MVIP Multi-Chassis boards where **f** is the minor switch block, **ss** is the stream and **tt** is the timeslot.

**Response**
None

**Comments**

The MVIP line boards have a single minor switch block. Streams supported are 0x00-0x11. The information returned for streams 0x00-0x07 will be identical to that for streams 0x08-0x0F. Stream 0x10 is used to connect to the analog ports and stream 0x11 is used to connect to the DSP. The values **cmhi, cmlo, and data** are read from registers in the FMIC chip on the board.

The MVIP multi-chassis board has three minor switch blocks. The first two of these form the MC1 connection. The third block is used for MVIP to MVIP connections and conferencing. The streams will be between 0x00 and 0x13 with the first 16 mapping into the MVIP bus and streams 0x10-0x13 being used for the connections to either the MC1 bus or the conferencing chips. The MC1 streams actually consist of two streams running at 4.096 MHz rather than four streams at 2.048, but for purposes of addressing the FMIC, the timeslots on the two streams are interleaved.

Information is returned from the board in a message of type 16, sub-type 4. The contents of the connection memory can be determined from the message which is returned in the string. This message takes the form:

"QDssttwwwpp" for MVIP line boards and "QDfssttwwwpp" for the Multi-Chassis board

where **f** is the minor switch block, **sstt** is the stream and timeslot, **www** is the 12 bits corresponding to the connection memory high and low bytes encoded as hexadecimal and pp is the data encoded as hexadecimal.

# QUERY_OUTPUT

**command: QUERY_OUTPUT**
**device number:** the device handle for the XDS board to receive the command
**parameters: &output_parms**

```
struct output_parms  {
int output_stream;          specifies the output stream number
int output_timeslot;        specifies the output timeslot
int mode;                   specifies the output mode, 0 - disabled, 1 - pattern mode
                            2 - connect mode
int input_stream;           receives the input stream number if in connect mode
int input_timeslot          receives the input timeslot if in connect mode
int message                 receives pattern data if in pattern mode
};
```

**Applicable Boards**
All XDS MVIP Line boards and the XDS MVIP Multi-Chassis Board. A variant is available for the XDS Switch Matrix Board.

**Purpose**
This command retrieves the mode of a specific switch block output and any data associated with its mode.

**Returns**
**SUCCESS**
**MVIP_INVALID_STREAM**
**MVIP_INVALID_TIMESLOT**
**MVIP_INVALID_MODE**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_PARAMETER**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**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.

# QUERY_SWITCH_CAPS

**command: QUERY_SWITCH_CAPS**
**device number:** the device handle for the XDS board to receive the command
**parameters: &capabilities_parms**

struct capabilities_parms {

| | |
|---|---|
| int size; | specifies the size of the structure in bytes |
| int 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 networks; | receives the number of network streams |
| int channels[]; | an array receiving the number of channels for each network stream |

}

**Applicable Boards**
All XDS MVIP boards

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

**Returns**
**SUCCESS**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**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.
It is possible for the **size** of the structure to be smaller than is needed to pass back information about all of the network connections. If this is the case, then only as much information as there is space for will be returned in the structure.

# RESET_SWITCH

**command: 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**
**SUCCESS**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**Message Sent**
"RA"

**Response**
The Line Boards and Multi-Chassis 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.

# SAMPLE_INPUT

**command: SAMPLE_INPUT**
**device number:** the device handle for the XDS board to receive the command
**parameters: &sample_parms**

```
struct sample_parms {
int input_stream;        specifies the switch block input stream number
int input_timeslot;      specifies the switch block input timeslot
int input_sample;        receives the byte currently asserted on the switch block input
};
```

**Applicable Boards**
XDS MVIP Line Boards and the XDS MVIP Multi-Chassis Board.

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

**Returns**
**SUCCESS**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_TIMESLOT**
**MVIP_INVALID_STREAM**
**MVIP_INVALID_PARAMETER**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**
**XDS_INVALID_BOARD**

**Message Sent**
"QSsstt"

**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.

# SET_OUTPUT

**command: SET_OUTPUT**
**device number:** the device handle for the XDS board to receive the command
**parameters: &output_parms**

struct output_parms {

| | |
|---|---|
| int output_stream; | specifies the switch block's output stream |
| int output_timeslot; | specifies the switch block's output timeslot |
| int mode; | specifies the mode in which to place the switch output<br>0x00 = disable_mode<br>0x01 = pattern_mode<br>0x02 = connect_mode |
| int input_stream; | specifies the input stream in the connect mode |
| int input_timeslot; | specifies the input timeslot in the connect mode |
| int message; | specifies the pattern value to assert in the pattern mode |

} ;

**Applicable Boards**
All XDS MVIP Line boards, the XDS MVIP Multi-Chassis Board, and the XDS Switch Matrix Board (the Switch Matrix Board does not have pattern output capability)

**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**
**SUCCESS**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_PARAMETER**
**MVIP_INVALID_TIMESLOT**
**MVIP_INVALID_STREAM**
**MVIP_INVALID_MODE**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**Message Sent**
"MOssttiiiimpp" for Line 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 **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 **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 **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 **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.

# SET_TRACE

**command: SET_TRACE**
**device number:** the device handle for the XDS board to receive the command
**parameters: &trace_parms**

```
struct trace_params {
int code;
int (*printf)(const char *,...)
} ;
```

**Applicable Boards**
Not supported

**Purpose**
This is an optional diagnostic command that is not supported by either the library or the boards.

**Returns**
**MVIP_INVALID_COMMAND**

**Message Sent**
none

**Response**
None

**Comments**
This command is not supported by the XDS driver.

# SET_VERIFY

**command: SET_VERIFY**
**device number:** the device handle for the XDS board to receive the command
**parameters: &verify**

| | |
|---|---|
| int verify; | Specifies whether to enable or disable verification, 0x0 disables verification, a non-zero value enables verification |

**Applicable Boards**
All XDS MVIP Boards

**Purpose**
This command enables or disables command-by command verification of all switch operations. If verification is enabled, low-level switch I/O operations are verified and the error MVIP_SWITCH_VERIFY_ERROR is returned if an internal switch error is detected.

**Returns**
**SUCCESS**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_PARAMETER**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**Message Sent**
"SVv" for the XDS MVIP Line Boards and the XDS MVIP Multi-Chassis Board
"AE" or "AD" for the Switch Matrix Board

**Response**
The boards will respond with a message of type 2, subtype 3 with the msg_str equal to the message enabling the verify. No response is generated for disabling the verify.

**Comments**
All XDS MVIP boards will echo any commands when the verify is enabled. Commands will be returned preceded by an 'A'. Because the command message must be parsed before the verification message is sent, a period of several tens of milliseconds may elapse between the time a command is issued and a verification message is returned. Verification is disabled by default.

# TRISTATE_SWITCH

**command: TRISTATE_SWITCH**
**device number:** the device handle for the XDS board to receive the command
**parameters: &tristate**

| | |
|---|---|
| int tristate; | a value specifying whether to enable or disable the switch block if the value is 0x1 the block is tri-stated, if 0x0, the block is enabled |

**Applicable Boards**
All XDS MVIP boards.

**Purpose**
This command enables or disables the entire switch block with respect to the MVIP bus. This command can be used to tri-state the output buffers to the MVIP bus for diagnostic purposes. This command has no effect on local or network busses.

**Returns**
**SUCCESS**
**MVIP_MISSING_PARAMETERS**
**MVIP_INVALID_PARAMETER**
**XDS_NO_BOARD**
**XDS_NO_RESPONSE**

**Message Sent**
"MTD" to disable output to the MVIP bus (tristate enabled)
"MTE" to enable output to the MVIP bus (tristate disabled)
For the Switch Matrix, a set direction message "SDXXXXXXXX" is sent to tristate the board. Appropriate direction information is sent to disable the tristate.

**Response**
None

**Comments**
On the MVIP Line boards and Multi-Chassis Boards, the tristate function is carried out using the FMIC chips. On the Switch Matrix Board, the tristate function is accomplished by using the direction control logic. The driver keeps a table containing the direction information for use by this command.

# 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/XP Implementation**

The specific implementation for Windows NT, Windows 2000, and Windows XP 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/XP 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.

# MVIP-95 Software
# 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/XP Implementation

The specific implementation for Windows NT, Windows 2000, and Windows XP 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/XP 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.

# CT-BUS Software
# 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 H.100/H.110 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 & XDS Function Command Codes

This page was intentionally left blank.

# MVIP-90 Command Codes

Standard MVIP-90 Commands

| | | |
|---|---|---|
| **RESET_SWITCH** | **0x00** | resets switch block to known state |
| **QUERY_SWITCH_CAPS** | **0x01** | returns switch block capabilities |
| **SET_OUTPUT** | **0x10** | makes & breaks switch connections |
| **QUERY_OUTPUT** | **0x11** | returns state of a switch output |
| **SAMPLE_INPUT** | **0x12** | returns data of switch input |
| | | |
| **CONFIG_CLOCK** | **0x20** | configures MVIP clocking options |
| **MC1_CONFIG_CLOCK** | **0x21** | configures the MC1 left & right clocks |
| **MC1_SET_8KREF_CLOCK** | **0x22** | specifies the source of 8KREF |
| **MC1_SET_SEC8K_CLOCK** | **0x23** | specifies the source of SEC8K |
| | | |
| **DUMP_SWITCH** | **0x70** | returns contents of switch component |
| **SET_TRACE** | **0x71** | enables printing of diagnostic info |
| **TRISTATE_SWITCH** | **0x72** | enables/disables MVIP switch block |
| **SET_VERIFY** | **0x73** | enables verification |

# MVIP95 Command Codes

MVIP95 Standard Commands

| | | |
|---|---|---|
| **MVIP95_CMD_RESET_SWITCH** | **0x101** | resets switch block to known state |
| **MVIP95_CMD_QUERY_SWITCH_CAPS** | **0x102** | returns switch block capabilities |
| **MVIP95_CMD_SET_OUTPUT** | **0x103** | makes & breaks switch connections |
| **MVIP95_CMD_QUERY_OUTPUT** | **0x104** | returns state of a switch output |
| **MVIP95_CMD_SAMPLE_INPUT** | **0x105** | returns data of switch input |
| **MVIP95_CMD_CONFIG_STREAM_SPEED** | **0x11A** | configure speed for HMVIP streams |
| **MVIP95_CMD_QUERY_STREAM_SPEED** | **0x11B** | query speed for an HMVIP stream |
| **MVIP95_CMD_CONFIG_LOCAL_STREAM** | **0x1A0** | configure local device (stream**)** |
| **MVIP95_CMD_CONFIG_LOCAL_TIMESLOT** | **0x1A1** | configure local device (timeslot**)** |
| **MVIP95_CMD_QUERY_BOARD_INFO** | **0x1A2** | retrieve information about a board being controlled by device driver |
| **MVIP95_CMD_QUERY_DRIVER_INFO** | **0x1A3** | retrieve information about device driver |
| **MVIP95_CMD_QUERY_LOCAL_STREAM** | **0x1A4** | retrieve information about local device (stream**)** |
| **MVIP95_CMD_QUERY_LOCAL_TIMESLOT** | **0x1A5** | retrieve information about local device (timeslot**)** |

MVIP95 Clock Command Codes

| | | |
|---|---|---|
| **MVIP95_CMD_CONFIG_BOARD_CLOCK** | **0x111** | configure MVIP clocks |
| **MVIP95_CMD_CONFIG_SEC8K_CLOCK** | **0x112** | configure secondary 8KHz clock |
| **MVIP95_CMD_CONFIG_NETREF_CLOCK** | **0x113** | configure secondary NETREF clock |
| **MVIP95_CMD_QUERY_BOARD_CLOCK** | **0x114** | retrieve board clocking information |
| **MVIP95_CMD_CONFIG_8KREF_CLOCK** | **0x122** | configure MC1 8KHz clock signal |

# CT-BUS Command Codes

CT-BUS Standard Commands

| | | |
|---|---|---|
| **CTBUS_CMD_RESET_SWITCH** | **0x101** | resets switch block to known state |
| **CTBUS_CMD_QUERY_SWITCH_CAPS** | **0x102** | returns switch block capabilities |
| **CTBUS_CMD_SET_OUTPUT** | **0x103** | makes & breaks switch connections |
| **CTBUS_CMD_QUERY_OUTPUT** | **0x104** | returns state of a switch output |
| **CTBUS_CMD_SAMPLE_INPUT** | **0x105** | returns data of switch input |
| **CTBUS_CMD_CONFIG_STREAM_SPEED** | **0x11A** | configure speed for HMVIP streams |
| **CTBUS_CMD_QUERY_STREAM_SPEED** | **0x11B** | query speed for an HMVIP stream |
| **CTBUS_CMD_CONFIG_LOCAL_STREAM** | **0x1A0** | configure local device (stream**)** |
| **CTBUS_CMD_CONFIG_LOCAL_TIMESLOT** | **0x1A1** | configure local device (timeslot**)** |
| **CTBUS_CMD_QUERY_BOARD_INFO** | **0x1A2** | retrieve information about a board being controlled by device driver |
| **CTBUS_CMD_QUERY_DRIVER_INFO** | **0x1A3** | retrieve information about device driver |
| **CTBUS_CMD_QUERY_LOCAL_STREAM** | **0x1A4** | retrieve information about local device (stream**)** |
| **CTBUS_CMD_QUERY_LOCAL_TIMESLOT** | **0x1A5** | retrieve information about local device (timeslot**)** |

CT-BUS Clock Commands

| | | |
|---|---|---|
| **CTBUS_CMD_CONFIG_BOARD_CLOCK** | **0x111** | configure MVIP clocks |
| **CTBUS_CMD_CONFIG_SEC8K_CLOCK** | **0x112** | configure secondary 8KHz clock |
| **CTBUS_CMD_CONFIG_NETREF_CLOCK** | **0x113** | configure secondary NETREF clock |
| **CTBUS_CMD_QUERY_BOARD_CLOCK** | **0x114** | retrieve board clocking information |
| **CTBUS_CMD_QUERY_TIMING_REF** | **0x115** | retrieve board timing information |
| **CTBUS_CMD_CONFIG_8KREF_CLOCK** | **0x122** | configure MC1 8KHz clock signal |

# XDS Command Codes

| | | |
|---|---|---|
| Generic XDS Commands | | |
| **XDS_RESET_ALL** | **0x80** | issues an “RA” command to board |
| **XDS_ID** | **0x81** | returns board ID & version |
| **XDS_MSG_ON** | **0x82** | enables return messages |
| **XDS_MSG_OFF** | **0x83** | disables return messages |
| **XDS_MSG_SEND** | **0x84** | sends a message to the board |
| **XDS_MSG_RECEIVE** | **0x85** | returns received messages if any |
| **XDS_QUERY_RECEIVE** | **0x86** | returns query response if any |
| **XDS_SET_ENCODING** | **0x87** | sets encoding mode |
| XDS Multi-Chassis Commands | | |
| **XDS_MC1_SELECT** | **0x48** | enable MC1 bus transmit streams |
| Conference Commands | | |
| **MAKE_CONFERENCE** | **0x4B** | control conference resources |
| XDS Matrix-Board Commands | | |
| **XDS_MX_SET_DIRECTION** | **0x50** | set MVIP bus direction |
| **XDS_MX_SEND_DTMF** | **0x52** | send DTMF tone string to MVIP bus |
| XDS Line Board Commands | | |
| **XDS_MVIP_CONNECT** | **0x60** | seize a port & enable local audio |
| **XDS_RLS** | **0x61** | release a port & disable local audio |
| **XDS_CPTONES** | **0x62** | plays call progress tones to a port |
| **XDS_LISTEN_DTMF** | **0x63** | sets a port to listen for DTMF tones |
| **XDS_SEND_DTMF** | **0x64** | plays a DTMF tone string to a port |
| **XDS_HOOKFLASH** | **0x65** | generates a hookflash on a port |
| **XDS_MWI** | **0x66** | controls MWI for a station port |
| **XDS_RING** | **0x67** | generates ringing on a station port |
| **XDS_SEIZE** | **0x68** | seizes a port if in the idle state |
| **XDS_SET_PROTOCOL** | **0x69** | sets DID protocol for a DID or E&M |
| **XDS_SET_TYPE** | **0x6A** | sets port type |
| **XDS_RESET_DSP** | **0x6B** | resets the DSP on a line board |
| **XDS_DSP_VERSION** | **0x6C** | returns DSP version for a line board |
| **XDS_LINE_STATE** | **0x6D** | returns the line state for a port |

This page was intentionally left blank.

# MVIP-Related & XDS Function Return Codes

This page was intentionally left blank.

# MVIP-90 Return 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 Return 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 Return 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 Return Codes

| | | |
|---|---|---|
| **BRD_ERROR** | **-1** | **Board Error** |
| **ILL_PORT** | **1** | **Illegal Port** |
| **ILL_SLOT** | **2** | **Illegal Timeslot** |
| **ILL_ARG** | **3** | **Illegal Argument** |
| **ILL_HAND** | **4** | **Illegal Conference Handle** |
| **ILL_ATTEN** | **5** | **Illegal Attenuation** |
| **ILL_THRESHOLD** | **6** | **Illegal Threshold** |
| **WRONG_QUERY** | **7** | **Wrong Query** |
| **WRONG_BOARD** | **8** | **Wrong Board** |
| **NO_UPDATE** | **9** | **No Update** |
| **ILL_CCA** | **10** | **Illegal Conference Handle** |
| **ILL_MODE** | **11** | **Illegal "mode" value** |
| **ILL_TYPE** | **12** | **Illegal "type" (BRI) value** |
| **ILL_FEATURE** | **13** | **Illegal "feature" (BRI) value** |
| **ILL_CAUSE** | **14** | **Illegal "cause" (BRI) value** |
| **ILL_REFERENCE** | **15** | **Illegal "reference" (BRI) value** |
| **ILL_PROGRESS** | **16** | **Illegal "progress" (BRI) value** |
| **ILL_SIGNAL** | **17** | **Illegal "signal" (BRI) value** |
| **ILL_THRES** | **18** | **Illegal threshold value** |
| **ILL_PORT_TYPE** | **19** | **Illegal Port Type** |

# XDS IOCTL Return Codes

| | | |
|---|---|---|
| **SUCCESS** | **0** | returned successfully |

## UNIX

| | | |
|---|---|---|
| **XDS_MSG_AVAILABLE** | **0** | message available |
| **XDS_BOARD_NOT_PRESENT** | **1** | XDS board not present |
| **XDS_NO_MSG** | **1** | no messages on queue |
| **XDS_BOARD_NOT_RESPONDING** | **2** | XDS board is not responding |
| **XDS_DPRAM_BAD_WRITE** | **2** | attempt to write to first 256 bytes of boards DPRAM |
| **XDS_DPRAM_READ_OFF** | **2** | attempt to read at an offset before the beginning of the board |
| **XDS_DPRAM_WRITE_LIMIT** | **3** | attempt to write beyond the 2k limit |
| **XDS_DPRAM_READ_LIMIT** | **3** | attempt to read beyond the 2k limit |
| **XDS_BAD_COMMAND** | **4** | non supported ioctl command |

## Windows NT/2000

| | | |
|---|---|---|
| **XDS_ERR_IOCTL_XMT** | **256** | XMT command failed |
| **XDS_ERR_IOCTL_RCV** | **257** | RCV command failed |
| **XDS_ERR_IOCTL_RCV_QUERY** | **258** | RCV_QUERY command failed |
| **XDS_ERR_IOCTL_WRITE_DPRAM** | **259** | WRITE_DPRAM command fail |
| **XDS_ERR_IOCTL_READ_DPRAM** | **260** | READ_DPRAM command |
| **XDS_ERR_IOCTL_INVALID_COMMAND** | **261** | Invalid ioctl() command |
| **XDS_ERR_IOCTL_RESET** | **262** | XDS_RESET error |
| **XDS_ERR_GET_BOARD_INFO** | **263** | XDS_GET_BOARD_INFO error |
| **XDS_ERR_GET_BUS_DEVICE_NUM** | **264** | XDS_GET_BUS_DEVICE_NUM error |
| **XDS_ERR_IOCTL_SLEEP** | **265** | XDS_SLEEP error |
| **XDS_ERR_IOCTL_RESUME** | **266** | XDS_RESUME error |
| **XDS_ERR_QUEUE_USER_MSG** | **267** | XDS_QUEUE_USER_MSG error |
| **XDS_ERR_IOCTRL_BUSY** | **268** | board transmit flag is not clear (error) |
| **XDS_ERR_IOCTRL_SYSTEM** | **269** | Windows system timer not available (error) |

This page was intentionally left blank.