XDS Windows NT H.100 / SCSA Driver Package Reference Manual

Driver Version 4.6
April 2004

American Tel-A-Systems, Inc.

Contents

XDS Windows NT SCSA / H.100 Driver Reference Manual

American Tel-A-System, Inc.
608-838-4194
4800 Curtin Drive, McFarland, WI 53558, USA
http://www.amtelco.com/
251M015N

Driver Package Software Installation and Removal

This page was intentionally left blank.

1.0 Driver Package Installation

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

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

Each XDS ISA board has two rotary switches labeled SW1 and SW2 and a set of jumper blocks labeled JW5. SW1 is used to select which 2K portion of memory within the 32K segment of shared memory mapped into the PC’s address space the XDS board will use. **Each board** must have a unique setting of this switch. SW2 is used to select the addresses of the I/O ports used by the boards. This switch must be set the same for **all** XDS ISA boards in the system. The jumper block labeled JW5 is used to select the hardware interrupt shared by all XDS boards. This jumper should be set the same for **all** XDS ISA boards in the system. It is important that the I/O addresses and interrupt selected not be used by any other hardware in the computer. Collectively, the XDS ISA boards will occupy a 32K block of memory. The starting address of this block should be chosen so that it does not conflict with other hardware in the system, such as video boards or disk controllers. For details on the hardware interface, consult the appropriate XDS technical manual.

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

The XDS Windows NT MVIP Driver Package comes in the form of one CDROM disc (Amtelco part number 251CD003). The driver package contains the driver, an installation utility, test and demonstration programs, and the source code for all of the included programs. To install the driver, insert the CDROM. If the driver package installation does not start automatically, it will need to run manually. This can be done by running *sc_nt_46.exe* (on the CDROM) from '**RUN'** from the start menu, or by double-clicking on the icon for it in Windows Explorer. The WISE Install Wizard will un-package the install contents and guide the user through the rest of the procedure. This will create the all the necessary directories, copy files to the hard drive, make the necessary modifications to the Windows NT registry, and add the *Amtelco SCSA Technology* folder to the Start Menu Programs. When setup has successfully completed, a new window will appear on the desktop containing the XDS program icons. These will be labeled "XdsInst", "XdsUtil", "Signal_Test", and "Sig_Util". Note that setup does NOT install the low-level device driver for the boards. That procedure is described in the next section, section 2.0 (Low-level Driver Installation).

2.0 Low-level Driver Installation

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

<drive>:\Program Files\Amtelco\SCSA\Bin\Intel

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

If only PCI boards will be used in this system, simply click on the “Go” button.

If there are any ISA boards to be used, the user must now set the required parameters for the ISA portion of the device driver. These are: the RAM base address, the SW2 switch setting to select the I/O address, and the JW5 IRQ interrupt selection jumper setting. The parameter values will be saved in the Windows NT Registry, and should only be modified using this installation program. When finished selecting the ISA board parameters, check the “Install ISA” option box.

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

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

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

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

3.0 Low-level Driver Removal

The program **XdsInst** is also used to unload or uninstall the XDS device driver. The user may unload (stop) a selected driver or uninstall (remove) the driver with this utility.

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

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

4.0 Low-level Driver Control

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

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

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

5.0 Driver Package Removal

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

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

Driver Package
Programs and Source Code

This page was intentionally left blank.

1.0 XDS Source Code Description

All of the source code used to build the programs, DLLs, and driver has been included for the user's convenience. If any or all of the code is "re-used", the American Tel-A-Systems, Inc. copyright information must be included with it. All of the project workspaces for this release package have a pre-processor define ("XDS_SCSA") 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 NT/2000 DDK were used to compile and build the low-level driver (xds.sys).

2.0 Tests / Utilities

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

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 display 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.

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.

Driver IOCTL Command Line Test
The test program **test_drv** is a simple program, written in ‘C’, that demonstrates how to make driver IOCTL calls. It is a text-based command line application that makes IOCTL calls directly to the driver. The syntax is “**test_drv n”**, where **n** is the number of an installed XDS board. The program first displays any messages that might be already on the board’s queue(s). 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.

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 Test

The test program test_dll.exe is an example of how the XDS H.100 SCSA-based Native DLL can be linked to a program and tested. All of the functions in this program are included in the XdsLibSc DLL. The syntax is "**test_dll n"**, where **n** is the number of an installed XDS board. 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.

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.

T1E1LedDemo
The T1E1LedDemo program is a demo / utility for the user, which monitors the span status for a given T1/E1 board. If more than one T1/E1 board is present in the system, the user will need to select the board number of the desired board to monitor. This is done when the program is executed by selecting from the drop-list the board number.

XDS_T1E1_Config
T1/E1 board configuration utility. Please refer to section 3, “XDS T1 / E1 Configuration Utility Program”, for a description of this utility.

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 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> <segment> <board number (decimal)>
```

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

```
wn386dlc 257H001.HEX c 16
```

will flash the firmware file, 257H001.hex, to the control processor onto board number 16.

4.0 DLL Description

XdsLibSc DLL
An “XDS” DLL (XdsLibSc.dll) has been provided to access XDS native board functions. These include proprietary functions for use with XDS boards. Many of the applications in this package use this DLL. When creating a new application, be sure to link in XdsLibSc.lib in the project workspace. Details of the functions included in this library may be found in the document *XDS SCSA Library Reference Manual, 251M024.*

5.0 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 -

\SCSA\bin\intel	- executables, DLLs, driver, and downloader

Source code directories -

\SCSA\source\Downloader	- wn386dlc downloader program
\SCSA\source\Include	- include (header) files
\SCSA\source\Driver	- xds.sys low-level driver
\SCSA\source\Lib	- library files for Intel x86 processors
\SCSA\source\Mc3_Fiber_Test	- MC3 fiber ring test source code
\SCSA\source\Shared	- shared source code directory
\SCSA\source\Sig_Util	- Sig_Util application source code
\SCSA\source\Sig_Util2	- Sig_Util2 application source code
\SCSA\source\Signal_Test	- Signal_Test application source code
\SCSA\source\Station_Config	- station_config (utility) application
\SCSA\source\Test_dll	- xdslibmv.dll test
\SCSA\source\Test_isa_drv	- ISA driver test
\SCSA\source\Test_drv	- driver IOCTL test
\SCSA\source\Wise	- Wise installation project file
\SCSA\source\XDS_BRI_Config	- xds_bri_config (utility) application
\SCSA\source\XdsInst	- xdsinst driver installation program
\SCSA\source\XdsLibSc	- xdslibsc.dll (XDS SCSA native functions)

\SCSA\source\XdsPciRes	- xdspcires (utility) application source code
\SCSA\source\XdsUtil	- xdsutil (utility) application source code
\SCSA\source\Xds_T1E1_Config	- T1/E1 configuration (utility) application source code
\SCSA\source\T1E1LedDemo	- T1/E1 span-status (demo) application source code

This page was intentionally left blank.

XDS Windows NT
Driver IOCTL Description

This page was intentionally left blank.

Overview

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

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

The 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_BOARD_ID	- obtain the board ID from DPRAM
XDS_RESET	- reset specified device (ISA High Density Line Boards, ISA BRI, all H.100, and all H.110 boards)
XDS_HR_ACK	- hardware removal (Hot-swap/H.110 only)
XDS_GET_BUS_DEVICE_NUM	- obtain the PCI bus and slot number for a given XDS PCI board

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

For ISA boards, this number corresponds to the SW1 switch setting of the board. These numbers will range from 0-15.

For PCI/H.100 boards, this number will correspond to the PCI device number. These numbers will range from 16-31.

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.

Application Interface

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

CreateFile

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

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

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

CloseHandle

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

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

DevIoControl

The DevIoControl call takes the form:

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

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

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

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_BOARD_ID

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_BOARD_ID,	board ID command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
pData,	pointer to output structure
sizeof(XDS_MSG),	length of output structure
&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 obtain the ID of a specified board.

Returns
The function will return the following codes:

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

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

XDS_RESET

BOOL DevIoControl(	
hdriver,	device handle
(DWORD)XDS_RESET,	hardware reset command
pData,	pointer to input structure
sizeof(XDS_MSG),	length of input structure
NULL,	pointer to output structure
0,	length of output structure
&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 and cPCI H.110 boards only.

This page was intentionally left blank.