# XDS Windows 2000/XP H.100/SCSA-Based Driver Reference Manual

**October 2005**

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

**257M018H ©**

This page was intentionally left blank.

# Contents

# XDS Windows 2000/XP PCI (H.100) / ISA SCSA
# 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/
**257M018H**

# ISA / PCI Driver
# And Package Software
# Installation And Removal

This page was intentionally left blank.

**Driver Package Contents** -

The XDS Windows 2000/XP H.100 (SCSA-based) Driver package comes in the form of a CD-ROM disc (Amtelco P/N 257CD004) 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), a "WISE Wizard" 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_pci.sys) into the \Program Files\Amtelco\SCSA\ directory along with the .inf file.

# 1.0 Hardware Installation

ISA -

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.

PCI -
Each XDS H.100 board uses 8K of memory and comes in the PCI 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 PCI board(s) and one ISA interrupt available for the ISA 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 PCI 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.

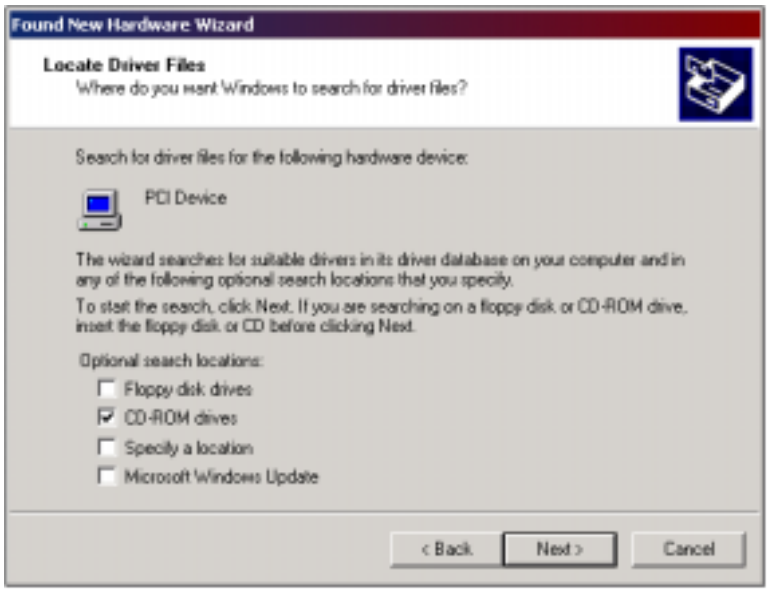


**Figure 1.3**

The next window should look like Figure 1.4 when "xds_2000_pci.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.

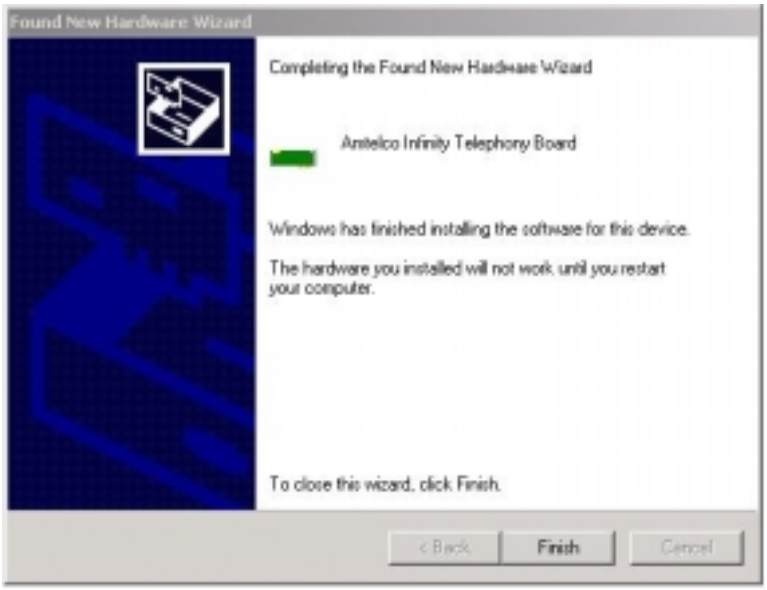


**Figure 1.5**

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.

The user may be required to restart the machine after the driver has been installed. If Windows prompts you to do this, it will need to be done before the hardware can be used in order to obtain the appropriate hardware resources for the board(s).

When Windows 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.100 board will appear under the Computer Telephony Device class (Figure 2.0).

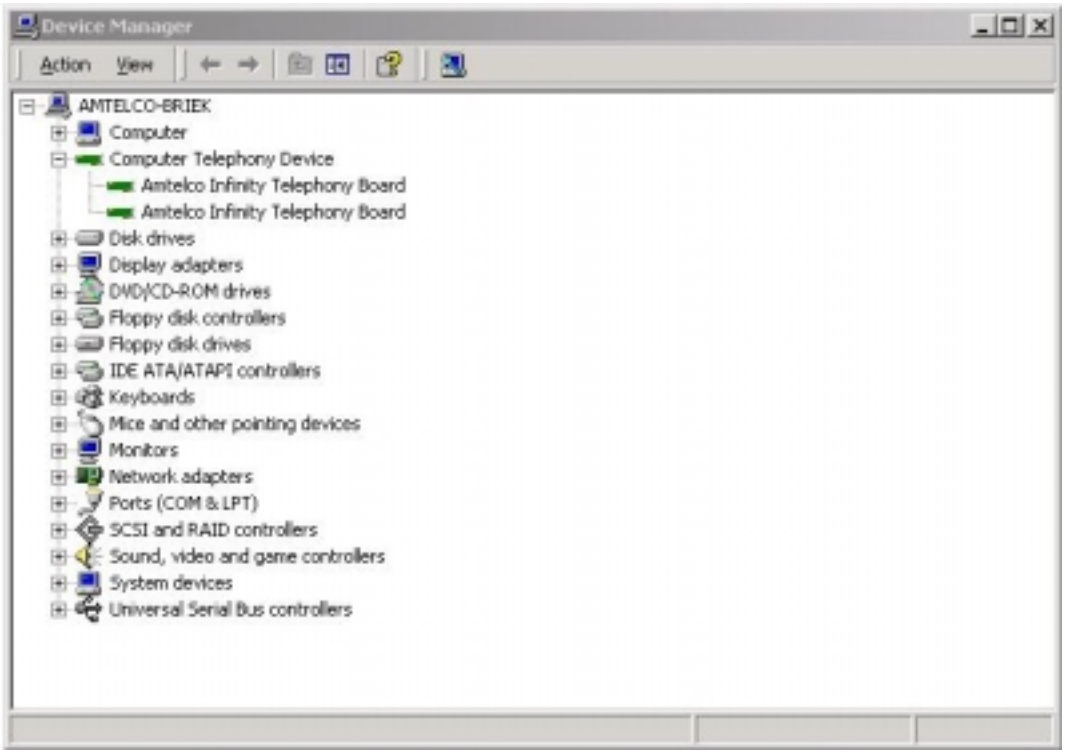


**Figure 2.0**

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_PCI

If problems occur with the driver, they will be identified in the **Event Viewer** with the *Event Source* "XDS_2000_PCI" 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, 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 ISA Driver Installation

If the user plans to use XDS ISA boards in the system, they will need to install the ISA low-level driver using the **XdsInst_Isa** utility. The default path for this program is: <drive>:\Program Files\Amtelco\SCSA\Bin\Intel

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 registry, and should only be modified using this installation program.

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 list-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 viewed by:

1) From the Windows **Programs** menu, open "Administrative Tools".
2) Open the "Computer Management"
3) Select the "Event Viewer" under the "system tools".
4) Events should appear, by default, in order from newest to oldest under "Applications".

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

NOTICE: There is only one pre-built program that will communicate with XDS ISA board in this driver package release. It is named Test_Isa_Drv.exe, and is located in the ..\Bin\Intel directory. The source code will give the user an idea of what they will need to do to modify the other programs to work with their ISA board(s).

# 5.0 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.

2) If the ISA driver was installed on the system, remove it. The can be done by executing "XdsInst_Isa", then selecting the **Unload** and **Uninstall** check-boxes. Now click on the "Go" button.

3) If the PCI driver was installed on the system, follow the steps below, starting with step 4a.

**4)** 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.100/SCSA-based Driver Pkg"**.

**4a**) 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**

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

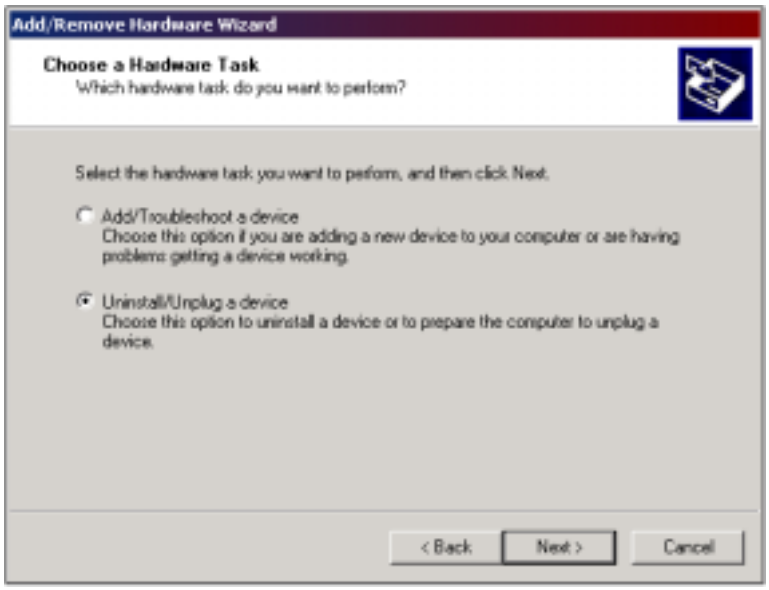


**Figure 3.1**

**4c**) Then, select **Uninstall a device** and click on **Next >** in the next window (like the one in Figure 3.2).

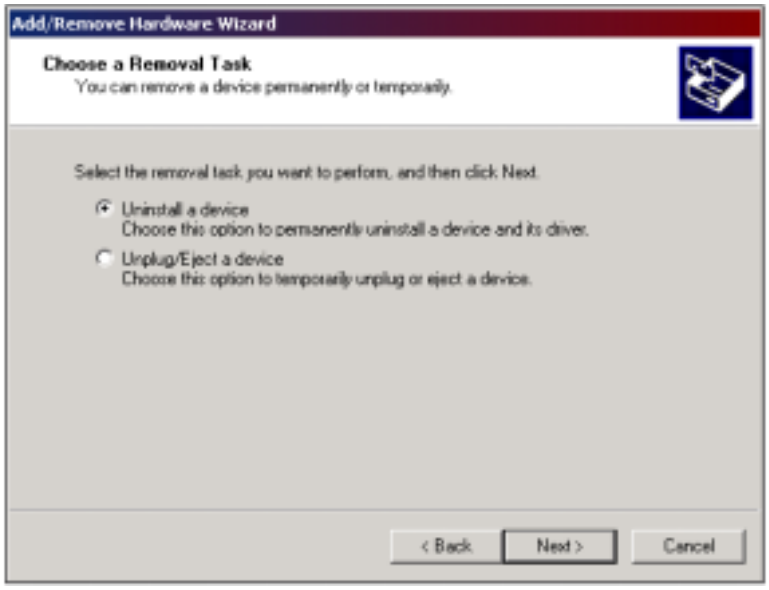


**Figure 3.2**

**4d)** You will now need to select the **Amtelco Infinity Telephony Board(s)** from the device list (as pictured in Figure 3.3) and click **Next >**.

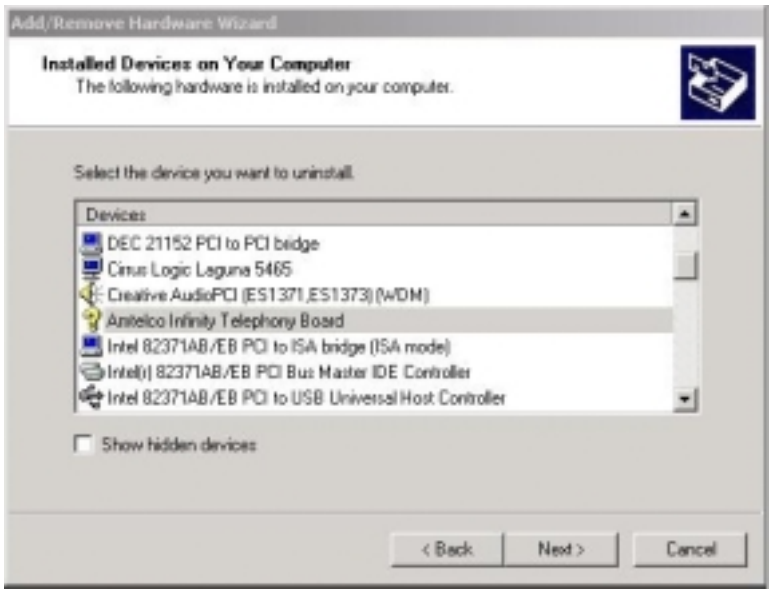


**Figure 3.3**

**4e**) Select **Yes, I want to uninstall this device** option from the following window (Figure 3.4) and click on **Next >**.

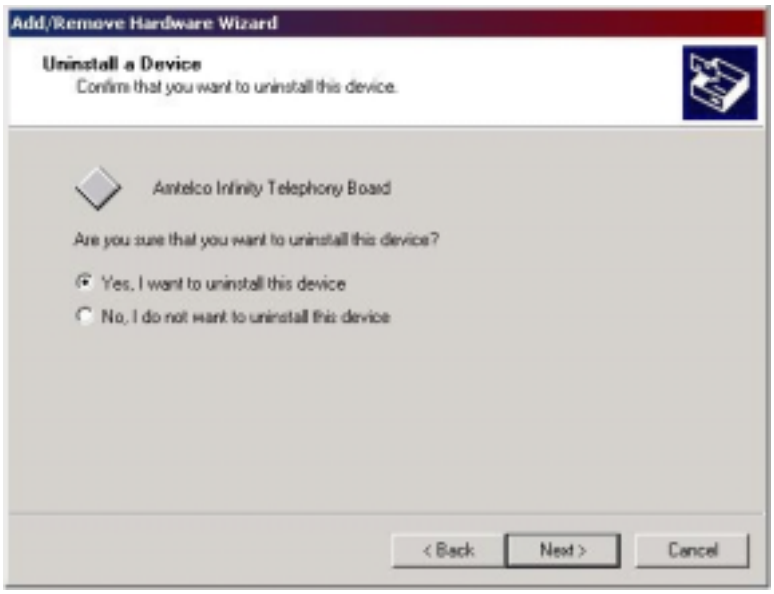


**Figure 3.4**

**4f**) Click on **Finish >** button in the last window (Figure 3.5).


**Figure 3.5**

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

2) 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 define ("XDS_SCSA" and "XDS_2000_PCI") 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 drivers (xds_2000_pci.sys and xds_2000_isa.sys).

# 2.0 Tests / Utilities

Several XDS demonstration applications are included with this package and may be found in the **..\bin\intel** directory. These are included to give the user an understanding of how the product works.

**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.

## PCI Driver Command Line Test

The test program **test_pci_drv** is a simple program, written in 'C', that demonstrates how to make PCI driver IOCTL calls. It is a text-based command line application that makes IOCTL calls directly to the PCI driver. The syntax is "**test_pci_drv n"**, where **n** is the number of an installed XDS board, or just "**test_pci_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.

## ISA Driver Command Line Test

The test program **test_isa_drv** is a simple program that demonstrates how to make ISA driver IOCTL calls. It is a text-based command line application that makes IOCTL calls directly to the ISA driver. The syntax is "**test_isa_drv n"**, where **n** is the SW1 setting 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, and the user should keep the design philosophy in mind when writing ones own application.

**Sig_Util2** is a Graphical User Interface that 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.

**signal_test** is a program that 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, 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.

## 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_Board_Dump

The Xds_Board_Dump utility is a command line utility that will dump the contents of the XDS board's dual-ported RAM to a text file (named *xdsdump.log*). This will be useful in trouble-shooting the board if there is a problem detected. The syntax is "**Xds_Board_Dump n"**, where **n** is the number of a specific XDS board, or just "**Xds_Board_Dump**", if only on XDS board is installed.

## 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 the program starts, 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 may 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 (i.e.: 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>  <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 XDS Voice Playback Demo Programs

Several XDS voice playback demonstration applications are included with this package and may be found in the **..\Bin\Intel\voice playback** directory. Included also are some sample voice files that are used by the demos. These are included to give the user an understanding of how the product works.

**Not all boards support this feature, so please check your board's reference manual to see if this feature is on your board and that it has the correct firmware.**

**As of this release the only audio formats supported for playback are raw A-law, raw μ-law, raw ADPCM, and A-law & μ-law in .wav format.**

### 4.1 xds_play_digits

xds_play_digits will play three introduction voice prompts for the user and then collect DTMF digits. When the user is done entering digits (eg: a phone number) the program will then play corresponding voice files back for each digit collected. It demonstrates to the user how to begin writing an application that will playback files on the XDS board that has voice playback support. This demo only uses XDS boards that have voice playback support. Digits will be collected as soon as they are detected by the XDS board that has voice playback support and will continue to be collected until the user has not pressed one in 5 seconds, or the user presses either a '*' or '#' digit.

### 4.2 xds_play_digits2

**xds_play_digits2** is an interactive XDS voice playback demo. It play three introduction voice prompts for the user and then collect DTMF digits. When the user is done entering digits (eg: a phone number) the program will then play corresponding voice files back for each digit collected. It demonstrates to the user how to begin writing an application that will playback files on the XDS board that has voice playback support and connect an XDS H.100 Station board to the XDS board that has voice playback support via the H.100 CT bus. The Station board will monitor audio on port 0. This demo uses an XDS board that has voice playback support and an XDS H.100 Station board. Digits will be collected as soon as they are detected by the XDS board that has voice playback support and will continue to be collected until the user has not pressed one in 5 seconds, or the user presses either a '*' or '#' digit.

### 4.3 xds_play_buffer

**xds_play_buffer** is a demonstration to the user how to play a file from memory (buffered) 3 times on the XDS board that has voice playback support. The user may stop the playback at any time during the loop by sending a DTMF digit to the XDS board that has voice playback support.

### 4.4 xds_play_buffer2

**xds_play_buffer2** is a demonstration to the user how to play a file from memory (buffered) 3 times on the XDS board that has voice playback support using an XDS H.100 Station to monitor the audio. The user may stop the playback at any time during the loop by sending a DTMF digit to the XDS board that has voice playback support.

### 4.5 xds_play_file_index

**xds_play_file_index** is a demonstration to the user how to play an indexed file list twice in a row on the XDS T1/E1 board. The user may stop the playback at any time during the loop by sending a DTMF digit to the XDS T1/E1 board.

### 4.5 xds_play_buffer_index

**xds_play_buffer_index** is a demonstration to the user how to play an indexed buffer list three times in a row on the XDS T1/E1 board. The user may stop the playback at any time during the loop by sending a DTMF digit to the XDS T1/E1 board.

# 5.0 XDS Voice Resource Demo Programs

Several XDS voice resource user demonstration applications are included with this package and may be found in the **..\Bin\Intel\voice resource** directory. Included also are some sample voice files that are used by the demos. These are included to give the user an understanding of how the product works.

**Not all boards support this feature, so please check your board's reference manual to see if this feature is on your board and that it has the correct firmware.**

**①As of this release the only audio formats supported for playback are 6kHz and 8kHz ADPCM, 8 bit A-law & μ-law in .wav format, and 16 bit Linear PCM in .wav format.**

**②As of this release the only audio formats supported for recording are 8kHz ADPCM, 8 bit A-law & μ-law in .wav format, and 16 bit Linear PCM in .wav format.**

### 5.1 xds_vr_custom_tone

**xds_vr_custom_tone** is an example of how to generate a custom tone on a voice resource channel. The program will prompt the user for the following options at runtime:

1) A voice resource channel to generate the tone on (0 – the max number of voice resource channels available).
2) A physical port / timeslot on the board to connect the output from the voice channel to.
3) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
4) The custom tone characteristics.

### 5.2 xds_vr_dial_dtmf

**xds_vr_dial_dtmf** is an example of how to dial a DTMF string on a voice resource channel. The program will prompt the user for the following options at runtime:

1) A voice resource channel to generate the tone on (0 – the max number of voice resource channels available).
2) A physical port / timeslot on the board to connect the output from the voice channel to.
3) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
4) The DTMF string to dial.

### 5.3 xds_vr_play_buffer

**xds_vr_play_buffer** is an example of how to playback a buffered (in user memory space) media file on a voice resource channel. The program will prompt the user for the following options at runtime:

1) A voice resource channel to playback on (0 – the max number of voice resource channels available).
2) A filename of the media file to be opened by the application, buffered, and played by the library.
3) The data format and file type of the media file.
4) A physical port / timeslot on the board to connect the output from the voice channel to.
5) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
6) Whether or not to enable full duplex on the voice resource channel.
7) Whether or not to enable DTMF detection on the voice resource channel.
8) Whether or not to stop on DTMF (if DTMF detection is enabled).
9) The DTMF digit(s) to stop on (if stop on DTMF is enabled).
10) The number of times to repeat playback. Options for this are 0 – 32767 or XDS_REPEAT_INFINITY (infinity (-1)).
11) Whether or not to increase the playback gain.
12) The playback gain (from 0 to 20 dB) to be used (if user chooses to increase the playback gain).

### 5.4 xds_vr_play_buffer_index

**xds_vr_play_buffer_index** is an example of how to playback a buffered (in user memory space) sample media file list on a voice resource channel. The program will prompt the user for the following options at runtime:

1) A voice resource channel to playback on (0 – the max number of voice resource channels available).
2) A physical port / timeslot on the board to connect the output from the voice channel to.
3) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
4) Whether or not to enable full duplex on the voice resource channel.
5) Whether or not to enable DTMF detection on the voice resource channel.
6) Whether or not to stop on DTMF (if DTMF detection is enabled).
7) The DTMF digit(s) to stop on (if stop on DTMF is enabled).
8) The number of times to repeat playback. Options for this are 0 – 32767 or XDS_REPEAT_INFINITY (infinity (-1)).
9) Whether or not to increase the playback gain.
10) The playback gain (from 0 to 20 dB) to be used (if user chooses to increase the playback gain).

### 5.5 xds_vr_play_file

**xds_vr_play_file** is an example of how to playback a media file on a voice resource channel. The program will prompt the user for the following options at runtime:

1) A voice resource channel to playback on (0 – the max number of voice resource channels available).
2) A filename of the media file to be opened and played by library.
3) The data format and file type of the media file.
4) A physical port / timeslot on the board to connect the output from the voice channel to.
5) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
6) Whether or not to enable full duplex on the voice resource channel.
7) Whether or not to enable DTMF detection on the voice resource channel.
8) Whether or not to stop on DTMF (if DTMF detection is enabled).
9) The DTMF digit(s) to stop on (if stop on DTMF is enabled).
   The number of times to repeat playback. Options for this are 0 – 32767 or XDS_REPEAT_INFINITY (infinity (-1)).
10) Whether or not to increase the playback gain.
11) The playback gain (from 0 to 20 dB) to be used (if user chooses to increase the playback gain).

### 5.6 xds_vr_play_file_index

**xds_vr_play_file_index** is an example of how to playback a sample list of media files on a voice resource channel. The files to be played back MUST all be of the same file type and data format while using the xds_vr_file_index_play() functions. The program will prompt the user for the following options at runtime:

1) A voice resource channel to playback on (0 – the max number of voice resource channels available).
2) A physical port / timeslot on the board to connect the output from the voice channel to.
3) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
4) Whether or not to enable full duplex on the voice resource channel.
5) Whether or not to enable DTMF detection on the voice resource channel.
6) Whether or not to stop on DTMF (if DTMF detection is enabled).
7) The DTMF digit(s) to stop on (if stop on DTMF is enabled).
   The number of times to repeat playback. Options for this are 0 – 32767 or XDS_REPEAT_INFINITY (infinity (-1)).
8) Whether or not to increase the playback gain.
9) The playback gain (from 0 to 20 dB) to be used (if user chooses to increase the playback gain).

### 5.7 xds_vr_play_file_gain_control

**xds_vr_play_file_gain_control** is an example of how to playback media file on a voice resource channel while controlling the gain during playback by using the '+' (to increase the gain) and '-' (to decrease the gain) buttons of the user's keyboard number pad. The program will prompt the user for the following options at runtime:

1) A voice resource channel to playback on (0 – the max number of voice resource channels available).
2) A filename of the media file to be opened and played by library.
3) The data format and file type of the media file.
4) A physical port / timeslot on the board to connect the output from the voice channel to.
   Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).

### 5.8 xds_vr_record_play_buffer

**xds_vr_record_play_buffer** is an example of how to record to a user buffer (in user memory space) on a voice resource channel. Once the recording is terminated, the demo program will then play back the contents of the user buffer on the same voice channel and physical port, so that they may check the contents. The program will prompt the user for the following options at runtime:

1) A voice resource channel to record and playback on (0 – the max number of voice resource channels available).
2) The data format of the voice data.
3) A physical port / timeslot on the board to connect the output from the voice channel to.
4) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
5) Whether or not to enable full duplex on the voice resource channel.
6) Whether or not to enable DTMF detection on the voice resource channel.
7) Whether or not to stop on DTMF (if DTMF detection is enabled).
   The DTMF digit(s) to stop on (if stop on DTMF is enabled).
8) Whether or not to increase the record gain.
9) The record gain (from 0 to 20 dB) to be used (if user chooses to increase the record gain).
10) Whether or not to suppress silence during the recording.
11) Whether or not to terminate recording on silence.
12) A period of silence to terminate on (from 1 to 229 seconds), if the user chooses the terminate on silence option.

### 5.9 xds_vr_record_play_file

**xds_vr_record_play_file** is an example of how to record to a media file on a voice resource channel. Once the recording is terminated, the demo program will then play back the contents of the media file on the same voice channel and physical port, so that they may check the contents. The program will prompt the user for the following options at runtime:

1) A voice resource channel to record and playback on (0 – the max number of voice resource channels available).
2) A filename of the media file to be opened and played by library.
3) The data format of the voice data.
4) A physical port / timeslot on the board to connect the output from the voice channel to.
5) Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).
6) Whether or not to enable full duplex on the voice resource channel.
7) Whether or not to enable DTMF detection on the voice resource channel.
8) Whether or not to stop on DTMF (if DTMF detection is enabled).
   The DTMF digit(s) to stop on (if stop on DTMF is enabled).
9) Whether or not to increase the record gain.
10) The record gain (from 0 to 20 dB) to be used (if user chooses to increase the record gain).
11) Whether or not to suppress silence during the recording.
12) Whether or not to terminate recording on silence.
13) A period of silence to terminate on (from 1 to 229 seconds), if the user chooses the terminate on silence option.

### 5.10 xds_vr_record_file_gain_control

**xds_vr_record_file_gain_control** is an example of how to record a media file on a voice resource channel while controlling the gain during recording by using the '+' (to increase the gain) and '-' (to decrease the gain) buttons of the user's keyboard number pad. The program will prompt the user for the following options at runtime:

1) A voice resource channel to record on (0 – the max number of voice resource channels available).
2) A filename of the media file to be recorded by library.
3) The data format and file type of the media file.
4) A physical port / timeslot on the board to connect the output from the voice channel to.
   Whether or not to monitor the audio on the physical port / timeslot (only if it is an XDS analog board).

The user may stop any of the demo programs at anytime by pressing the 'Esc' key on the computer's keyboard. Playback/record may also be stopped by DTMF if the option to stop on DTMF is enabled in that demo program.

### 5.11 xds_vr_demo

**xds_vr_demo** is an all-in-one GUI example demonstrating many of the XDS voice resource board functions, such as playing back and recording voice files and dialing DTMF.

1) First, select Open Voice Resource Channel from the File menu. From the drop-down list, the user will choose which *voice resource channel* on which XDS Voice Resource board they want to control.
2) Then the user will be prompted for a *physical port* to connect to. This is an analog port on the XDS Voice Resource board. This demo application will connect the stream and timeslot for a physical port for the user.
3) Once the voice resource channel has been selected and the port is connected, the user may select an option from the Function menu. These functions are described below.

Function > Play File:
The user may choose any supported media file ① on the host PC's hard disk drive to playback. If it is a supported .wav file, playback will begin immediately. If it is a supported raw media file ① (ie: 6 kHz or 8 kHz ADPCM), the user will then need to tell the application which data format to play back the file as.

Function > Play File Index:
This option will play back a list of pre-selected voice files on the voice resource channel that the user selected.

Function > Play Buffer:
The user may choose any supported media file ① on the host PC's hard disk drive to playback. If it is a supported .wav file, playback will begin immediately. If it is a supported raw media file ① (ie: 6 kHz or 8 kHz ADPCM), the user will then need to tell the application which data format to play back the file as. Playing from a buffer will open the voice file in user-space memory and then play it back.

Function > Play Digits:
The user may dial a DTMF string on a voice resource channel with this function. The user will enter the DTMF they wish to have dialed in the *Select digits to be played by channel* dialog window. Valid DTMF tones are 0-9, A-D, #, and *.

Function > Stop Play:
This option will stop playback on the channel that the user selects from the dialog window.

Function > Record File:
The user may record a voice file to hard disk by selecting this option. The user will be prompted to enter a filename (including the extension, i.e.: **.wav** – if a supported wave file) in the File name: field of the *Please select a filename for the recording* dialog window and will be able select the file type and data format in the Save as type: field.

### 5.12 xds_vr_read_dpram

**xds_vr_read_dpram** is a diagnostic utility which allows the user to view all of the voice resource DSP commands that were sent to and received from the DSP in addition to the transmit and receive mailbox head and tail.

# 6.0 XDS User Library / DLL Descriptions

**XdsLibSc DLL**

An "XDS" DLL (XdsLibSc.dll) has been provided to access XDS native board user functions. These include proprietary functions for use with XDS boards. Many of the applications in this software 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.*

**XdsVoiceLib DLL**

An "XDS" DLL (XdsVoiceLib.dll) has been provided to access XDS voice playback user functions. These include proprietary functions for use with XDS boards. All of the XDS voice playback applications in this software package use this DLL. When creating a new application, be sure to link in XdsVoiceLib.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.*

**XdsVrLib DLL**

An "XDS" DLL (XdsVrLib.dll) has been provided to access XDS Voice Resource user functions. These include proprietary functions for use with XDS boards. All of the XDS Voice Resource applications in this software package use this DLL. When creating a new application, be sure to link in XdsVrLib.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.*

# 7.0 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 |
| \SCSA\Bin\Intel\Voice Playback | - voice playback executables, sample .wav files, and XDS voice playback DLL |
| \SCSA\Bin\Intel\Voice Resource | - voice resource executables, sample .wav files, and XDS voice resource DLL |

Source code directories -

| | |
|---|---|
| \SCSA\Source\Downloader | - wn386dlc downloader program |
| \SCSA\Source\Include | - include (header) files |
| \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_pci_drv | - PCI driver test |
| \SCSA\Source\Wise | - Wise installation project file |
| \SCSA\Source\XDS_BRI_Config | - xds_bri_config (utility) application |
| \SCSA\Source\XDS_Class_Installer | - XDS WDM driver class installer |
| \SCSA\Source\XdsInst_Isa | - xdsinst_isa ISA driver installation program |
| \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 |
| \SCSA\Source\XdsWin2kRemove | - XDS low-level PCI driver removal script |
| \SCSA\Source\Xds_Board_Dump | - XDS diagnostic board memory dump tool |
| | |
| \SCSA\Source\Drivers\PciDriver | - xds_2000_pci.sys low-level driver |
| \SCSA\Source\Drivers\IsaDriver | - xds_2000_isa.sys low-level driver |
| | |
| \SCSA\Source\DLLs\XdsLibSc | - XDS SCSA native function library |
| \SCSA\Source\DLLs\XdsVoiceLib | - XDS voice playback library |
| \SCSA\Source\DLLs\XdsVrLib | - XDS Voice Resource library |

\SCSA\Source\Voice Playback\Xds_Play_Buffer
- XDS voice playback demo (from memory)
\SCSA\Source\Voice Playback\Xds_Play_Buffer_Index
- XDS voice playback demo (multiple prompts indexed from memory)
\SCSA\Source\Voice Playback\Xds_Play_Buffer2
- XDS voice playback demo (from memory)
\SCSA\Source\Voice Playback\Xds_Play_Digits
- XDS interactive voice playback demo (plays A-Law / u-Law .wav files prompted by user input)
\SCSA\Source\Voice Playback\Xds_Play_Digits2
- XDS interactive voice playback demo (plays A-Law / u-Law .wav files prompted by user input)
\SCSA\Source\Voice Playback\Xds_Play_File_Index
- XDS voice playback demo (multiple prompts indexed from A-Law / u-Law .wav files)

\SCSA\Source\Voice Resource\Xds_Vr_Custom_Tone
- XDS voice resource demo (generates a custom user tone)
\SCSA\Source\Voice Resource\Xds_Vr_Demo
- XDS GUI voice resource demo (plays and records voice data and dials DTMF)
\SCSA\Source\Voice Resource\Xds_Vr_Dial_Dtmf
- XDS voice resource demo (dials a DTMF string)
\SCSA\Source\Voice Resource\Xds_Vr_Play_Buffer
- XDS voice resource demo (plays back voice data from user memory space)
\SCSA\Source\Voice Resource\Xds_Vr_Play_Buffer_Index
- XDS voice resource demo (plays back indexed voice data from user memory space)
\SCSA\Source\Voice Resource\Xds_Vr_Play_File
- XDS voice resource demo (opens and plays back voice data from a supported media file)
\SCSA\Source\Voice Resource\Xds_Vr_Play_File_Index
- XDS voice resource demo (opens and plays back indexed voice data from supported media files)
\SCSA\Source\Voice Resource\Xds_Vr_Play_Gain_Control
- XDS voice resource demo (opens and plays back voice data from a supported media file while user can control the playback gain)
\SCSA\Source\Voice Resource\Xds_Vr_Record_Play_Buffer
- XDS voice resource demo (records voice data to user memory space and plays it back)
\SCSA\Source\Voice Resource\Xds_Vr_Record_Play_File
- XDS voice resource demo (records voice data to a media file and plays it back)
\SCSA\Source\Voice Resource\Xds_Vr_Record_Gain_Control
- XDS voice resource demo (records voice data to a media file while user can control the record gain)

# XDS T1 / E1
# Configuration Utility Program

This page was intentionally left blank.

## Overview:

This XDS utility program allows the user to set initial configuration parameter data for XDS T1 or E1 boards. It allows the user to configure a board, save the configuration data used in the EEPROM on the board, and save the configuration data used into a data file on the system's hard disk for future use (i.e.: on other board that the user may want to configure like the initial one). Before sending the configuration data to the board, the user will want to be very careful to review all selections to be sure that they are correct.

---

## Step 1 (Select the board to be configured) –

Select the board to be configured from the Board to configure drop-down list, and click on the "**Configure Board**" dialog button.

## Step 2 (Configuring each span) –

From the *Span Configuration* dialog window, select the desired choices for each span.

a) Span type
T1:
"CI" - Customer Interface
"NT" – network termination or CO interface
"Undef" - undefined for unused ports

E1:
"TE" - Terminal Equipment or user interface
"NT" - network termination or CO interface
"Undef" for undefined for unused ports.

b) Protocol layer
T1:
"AT&T Custom" – AT&T (Lucent) support
"DMS-100" - DMS-100 NI-1 support
"Layer 2" - Layer 2 support only
"Layer 3" - Layer 3 support (generic NI-1)
"NI-2" - NI-2 support
"Siemens" - Siemens CoreNet support

E1:
"Layer 2" - Layer 2 support only
"Layer 3" - Layer 3 support (generic NI-1)

c) Framing type
T1:
“D4” – DF Superframe
“ESF” – Extended SuperFrame

E1:
“CRC4” – Frame Aligned Signaling with additional error detection
“NonCRC4” – Frame Aligned Signaling

d) Suppression
T1:
“AMI” – Alternate Mark Inversion
“B8ZS” – Bipolar with 8-Zero Substitution

E1:
“AMI” – Alternate Mark Inversion
“HDB3” – High Density Bipolar with a maximum of 3 Zeros

e) Impedance or Line build
T1 (line build out):
“0” - DSX-1 (0-133ft.)/0dB CSU
“1” - DSX-1 (133-266ft.)
“2” - DSX-1 (266-399 ft.)
“3” - DSX-1 (399 to 533 ft.)
“4” - DSX-1 (533 to 655 ft.)
“5” – (-7.5) dB CSU
“6” – (-15) dB CSU
“7” – (-22.5) dB CSU

E1 (impedance):
“B” – 75 ohm impedance
“R” – 120 ohm impedance

f) Signaling mode
T1:
“N” - no signaling (used with NFAS)
“P” - Primary Rate ISDN
“R” - robbed-bit signaling

E1:
“C” – Channel Associated Signaling
“N” – no signaling
“P” – Primary Rate Access ISDN signaling
“S” – Signaling System 7, when the signaling channel is carried on another span and timeslot 16 is made available for use as an audio path
* Note that “N” and “S” are identical except for access to timeslot 16.

g) V 5.2 signaling detection (E1 boards only)
“Disable” – no V 5.2 signaling detection
“Enable” – use the V 5.2 signaling detection

## Step 3 (Configuring span channels) –

Click on the “**Span 0 channels**” dialog button in the Set channel options list to configure the channels on Span 0, click on the “**Span 1 channels**” dialog button in the Set channel options list to configure the channels on Span 1, and so forth…

a) Line type
T1:
“E” - E&M
“G” - ground start
“L” - loop start
“N” - none

E1:
“E” - E&M
“G” - ground start
“L” - loop start
“N” - none
“Q” - Q.421 signaling

b) Direction
“O” - FXO or CO side
“S” - the FXS or CPE side

c) Address protocol
“I” – immediate start
“W” – wink start

d) Number digits
Specifies the number of address digits to be expected on incoming calls

e) Directory number
This sets the default subscriber number on a Primary Rate ISDN/Access span. For channels on NT ports, this is the number that will be used as the default called directory number for calls originating on the port. For channels on ports defined as a CI/TE, this will be the number used as the calling party number. The directory number will be the calling number used for calls originating from the port. Subscriber numbers may be up to 15 digits in length.

Click the "**OK**" dialog button to save these settings and go back to the previous dialog window (*Span Configuration*), or click the "**Cancel**" dialog button to ignore any changes, if any were made, and go back to the previous dialog windows (*Span Configuration)*.

## Step 4 (Saving and using configured data) –

Once the configuration data is entered in all of the appropriate fields for each span and channel, the user may now save this data onto the board, setup the board with this configuration, or save it as a user-legible or binary file on the local disk drive.

Sending to board:
After ALL of the board configuration data is selected in all of the dialog boxes, review selections and click the "**Send to Board**" dialog button, in the *Span Configuration* dialog window. This will send all of the appropriate messages to the board and in-turn setup the board's run-time configuration tables with the user's entered selections.

Saving to EEPROM:
There is an EEPROM on the board, which can be used to save the configuration data in. You can do this by clicking on the "**Save to EEPROM**" dialog button, in the *Span Configuration* dialog window. The board will load this data from the EEPROM into the board's run-time configuration tables every time the board is restarted.

Saving configuration data to a file:
If there are multiple boards that will have the same configuration data, the user can save the configuration data to a file (in a local directory, on a hard disk drive) by clicking the "**Save to File**" dialog button, in the *Span Configuration* dialog window. This will save all of the configuration data used to a more user-friendly readable data file versus a binary image.

Configuring a board from a saved configuration file:
To configure a second (or other) board with data already saved (as a file), the user will select the board number of the board to be configure then click the "**Retrieve from File**" dialog button and give the correct file name.

Saving the board configuration data as a binary image:
The user may save the configuration data from a board as a binary image by clicking on the "**Save binary image**" dialog button in the main program dialog window *XDS T1/E1 Configuration Utility.* The board must be configured already, using the "**Save to File**" option in the *Span Configuration* dialog window.

Loading a binary image from a file in to a board:
The user may load configuration data from a binary image on the local hard drive by clicking on the "**Load binary image**" dialog button in the main program dialog window *XDS T1/E1 Configuration Utility.*

---

If you have any questions about the program, please contact Amtelco for technical support at (608) 838-4194, ex 168, or at xdssrvc@amtelco.com.

---

This page was intentionally left blank.

# XDS Windows 2000/XP PCI and ISA Driver IOCTL Description

This page was intentionally left blank.

# Overview

The XDS Windows 2000/XP Driver is designed to provide an interface between XDS boards and applications running under Windows 2000/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.

There are two interfaces used by the XDS boards. One is for the WDM plug-and-play PCI device driver, and the other is for the ISA device driver. 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 for all XDS boards:

| | |
|---|---|
| XMT | - transmit a standard message from an application to a board |
| RCV | - receive a message from a board on the response queue |
| RCV_QUERY | - receive a message from a board on 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_GET_BUS_DEVICE_NUM | - obtain the PCI bus and device number for a given XDS PCI board (PCI only) |
| XDS_QUEUE_USER_MSG | - place a board message on the receive message queue (PCI only) |
| XDS_GET_BOARD_INFO | - get board ID, version, number of ports, PCI bus and device number, board DPRAM size (in bytes), voice resource number of channels, voice resource module DPRAM size (in bytes), voice resource voice buffer size |

The DevIoControl supports the following commands for XDS boards that support voice playback (please check your board for compatibility before using these commands):

| | |
|---|---|
| VOICE_WRITE_DPRAM | - write a 1K block of data (audio) to the board's DPRAM |
| VOICE_RCV | - XDS voice message receive message function |
| VOICE_SET_EVENT | - enable signaling for voice playback messages |
| VOICE_UNSET_EVENT | - disable signaling for voice playback messages |

The DevIoControl supports the following commands for XDS boards that have a voice resource module (please check your board for compatibility before using these commands):

VOICE_RESOURCE_READ_DPRAM
- read data from the voice resource module's DPRAM

VOICE_RESOURCE_WRITE_DPRAM_PARAMETERS
- write parameter data to the voice resource module's DPRAM

VOICE_RESOURCE_WRITE_DPRAM_DATA
- write general data to the voice resource module's DPRAM

VOICE_RESOURCE_XMT
- send a command from the application to the voice resource module

VOICE_RESOURCE_RCV
- receive driver-to-application voice resource information messages

VOICE_RESOURCE_SET_EVENT
- enable signaling for voice resource driver-to-application information messages

VOICE_RESOURCE_UNSET_EVENT
- disable signaling for voice resource driver-to-application information messages

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.

The transmit command writes messages directly to the mailbox of the appropriate board. The driver places received messages on one of two commonly used queues. Acknowledgments, state change messages, error messages, etc… are passed through the *receive* queue. Query responses and version request responses are passed through a separate receive *query* queue. This queue is designed to help efficient programming by putting 'query' messages on a separate queue to cut down on access time by the application. Users will likely only access this queue when they expect a diagnostic message response from a message that they had sent. These messages may take a longer amount of time to generate a response, so therefore are on a separate queue. There are additional board-specific message queues for voice playback and voice resource messages respectively.

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. The voice playback and voice resource queues can both handle 127 messages. 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.

# PCI 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);

# ISA 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 device driver file name uses the symbolic link notation of "\\\\.\\XDS_2000_ISA".

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**, | (handle to device) |
| **DWORD** | **dwIoControlCode,** | **(**operation / command**)** |
| **LPVOID** | **lpInBuffer,** | **(**input data buffer**)** |
| **DWORD** | **nInBufferSize,** | **(**size of input data buffer) |
| **LPVOID** | **lpOutBuffer,** | **(**output data buffer) |
| **DWORD** | **nOutBufferSize,** | **(**size of output data buffer) |
| **LPDWORD** | **lpBytesReturned**, | (byte count) |
| **LPOVERLAPPED** | **lpOverlapped**); | (overlapped information) |

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,** | XDS device driver handle |
| **(DWORD)XMT,** | IOCTL low-level driver command |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | size of XDS_MSG data structure |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

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

**Purpose**

This command is used to 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 command 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,** | XDS device driver handle |
| **(DWORD)RCV,** | low-level driver command |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | size of XDS_MSG data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDS_MSG msg;

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

**Purpose**
This command is used to receive normal messages from XDS boards.

**Returns**
The command 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.

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.

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 Boards and T1/E1 boards 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,** | XDS device driver handle |
| **(DWORD)RCV_QUERY,** | low-level driver command |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | size of XDS_MSG data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDS_MSG msg;

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

**Purpose**

This command is used to receive version request responses and query responses, which are placed on the query response queue by the driver.

**Returns**

The command 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.

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.

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,                    XDS device driver handle
(DWORD)READ_DPRAM,          low-level driver command
&ram_info,                  pointer to XDS_DPRAM data structure
sizeof(XDS_DPRAM),          size of XDS_DPRAM data structure
NULL,                       (not used)
0,                          (not used)
&data_length,               pointer to number of bytes returned
NULL);                      (not used)

XDS_DPRAM                   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_DPRAM, *PXDS_DPRAM
```

**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 command 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,** | XDS device driver handle |
| **(DWORD)WRITE_DPRAM,** | low-level driver command |
| **&ram_info,** | pointer to XDS_DPRAM data structure |
| **sizeof(XDS_DPRAM),** | size of XDS_DPRAM data structure |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

| | |
|---|---|
| XDS_DPRAM | 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 a buffer with data to be written |
| } XDS_DPRAM, *PXDS_DPRAM | |

**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 command 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**
This command is included in the driver 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,** | XDS device driver handle |
| **(DWORD)XDS_RESET,** | low-level driver command |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | size of XDS_MSG data structure |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDS_MSG msg;

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

**Purpose**
This command is used to reset an entire board.

**Returns**
The command 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 the 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_GET_BUS_DEVICE_NUM

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD) XDS_GET_BUS_DEVICE_NUM,** | IOCTL low-level driver command |
| **&info,** | pointer to XDS_ID data structure (input) |
| **sizeof(XDSID),** | size of XDS_ID data structure |
| **&info,** | pointer to XDS_ID data structure (output) |
| **sizeof(XDSID),** | size of XDS_ID data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDSID info;

typedef struct xdsid
{

| | |
|---|---|
| char board_number; | the board number |
| char id[5]; | a character array containing the board ID string |
| char version[5]; | a character array containing the firmware version number |
| int number_ports; | the number of ports on the board |
| unsigned char pci_bus_number; | board's PCI bus number |
| unsigned char pci_device_number; | board's PCI device number |
| unsigned long board_memory_size; | XDS board DPRAM size |
| unsigned long vr_board_memory_size; | XDS Voice Resource board Blackfin DPRAM size (voice resource boards only) |
| unsigned char vr_board_number_channels; | number of voice channels available (voice resource boards only) |
| unsigned long vr_board_voice_buffer_size; | size of voice buffers (voice resource boards only) |

}XDSID, *PXDSID

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

**Returns**
The command 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. Although XDS_ID can provide this information, this command is left in for backwards compatibility.

# XDS_QUEUE_USER_MSG

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD)XDS_QUEUE_USER_MSG,** | IOCTL low-level driver command |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | length of message |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDS_MSG msg;
typedef struct{

| | |
|---|---|
| UCHAR board_number; | the board number |
| char msg[32]; | the ASCII text of message, NULL terminated |
| USHORT augTxRxLen; | length of Layer 3 message |
| UCHAR augTxRxMesg[260]; | body of Layer 3 message |

}XDS_MSG *PXDS_MSG;

**Purpose**
This command is used to 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 command 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,** | XDS device driver handle |
| **(DWORD) XDS_GET_BOARD_INFO,** | IOCTL low-level driver command |
| **&info,** | pointer to XDS_ID data structure (input) |
| **sizeof(XDSID),** | size of XDS_ID data structure |
| **&info,** | pointer to XDS_ID data structure (output) |
| **sizeof(XDSID),** | size of XDS_ID data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDSID info;

typedef struct xdsid
{

| | |
|---|---|
| char board_number; | the board number |
| char id[5]; | a character array containing the board ID string |
| char version[5]; | a character array containing the firmware version number |
| int number_ports; | the number of ports on the board |
| unsigned char pci_bus_number; | board's PCI bus number |
| unsigned char pci_device_number; | board's PCI device number |
| unsigned long board_memory_size; | XDS board DPRAM size |
| unsigned long vr_board_memory_size; | XDS Voice Resource board Blackfin DPRAM size (voice resource boards only) |
| unsigned char vr_board_number_channels; | number of voice channels available (voice resource boards only) |
| unsigned long vr_board_voice_buffer_size; | size of voice buffers (voice resource boards only) |

}XDSID, *PXDSID

**Purpose**

This command is used to obtain the ID, firmware version, number of physical/virtual ports, the PCI bus and device number, size of DPRAM, size of voice resource DPRAM (if available), number of voice resource channels (if available), and the size of voice resource buffers (if available) of a specified board.

**Returns**
The command 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 physical/virtual ports, PCI bus and device number, board DPRAM size, number of voice resource channels (if available), voice resource module DPRAM size (if available), and voice resource voice buffer (if available) size.

# READ_PLX_INT

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD) READ_PLX_INT,** | IOCTL low-level driver command |
| **&int_info,** | pointer to XDSINTCSR data structure (input) |
| **sizeof(XDSINTCSR),** | size of XDSINTCSR data structure |
| **&int_info,** | pointer to XDSINTCSR data structure (output) |
| **sizeof(XDSINTCSR),** | size of XDSINTCSR data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDSINTCSR int_info;

typedef struct xdsintcsr
{

| | |
|---|---|
| char plx_9030; | PLX 9030 or PLX 9080/9054 chip |
| char board_number; | the board number |
| long interrupt_enable; | interrupt enable register value |

} XDSINTCSR, *PXDSINTCSR

**Purpose**
This command is used to query the PLX's interrupt enable register value of a specified board.

**Returns**
The command 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 is useful in a diagnostic environment.

# VOICE_WRITE_DPRAM

**BOOL DevIoControl(**
**hdriver,** XDS device driver handle
**(DWORD)VOICE_WRITE_DPRAM,** IOCTL low-level driver command
**&data,** pointer to XDS_DPRAM data structure
**sizeof(XDS_DPRAM),** size of XDS_DPRAM data structure
**NULL,** (not used)
**0,** (not used)
**&data_length,** pointer to number of bytes returned
**NULL);** (not used)

XDS_BOARD_RAM data;

typedef struct xds_dpram{
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_DPRAM *PXDS_DPRAM

**Purpose**
This command is used to 1K blocks of data (audio) into the dual-ported RAM on the XDS voice playback board specified in board_number. It will write the data (audio) beginning at an offset of 0 each time.

**Returns**
The command 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 call is used to implement the audio playback feature of XDS boards that have voice playback firmware loaded. Please check your board manual to see if this feature is available.

# VOICE_RCV

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_RCV,** | IOCTL low-level driver command |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&msg,** | pointer to XDS_MSG data structure |
| **sizeof(XDS_MSG),** | size of XDS_MSG data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

XDS_MSG msg;

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

**Purpose**
This command is used to receive voice playback messages from XDS boards that support voice playback.

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

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_DATA_ERROR | no messages on voice playback message queue |

**Comments**
This command checks to see if there is any message on the voice playback message queue. Most of these messages will be “PT” and “PV” message responses. If the queue becomes full, a “FULL QUEUE” message is placed on the queue with the board_number for that message set to 255. 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.

Query version request messages, and normal XDS messages are returned on the message response and query response queues and read with the RCV and RCV_QUERY commands.

# VOICE_SET_EVENT

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_SET_EVENT,** | IOCTL low-level driver command |
| **&event_handle,** | pointer to an event HANDLE |
| **sizeof(HANDLE *),** | size of the event HANDLE |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

HANDLE event_handle;

**Purpose**
This command is used to enable the signaling mechanism in the driver for voice playback boards. When enabled, the driver will notify the calling function or application when a voice playback message arrives from an XDS board.

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

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_UNSUCCESSFUL | could not initialize event mechanism in driver |

**Comments**
To use the signaling mechanism, the application should create a user event handle and pass that handle value to the device driver using this IOCTL call. The application should then use some sort of wait mechanism, such as WaitForSingleObject() to monitor the receive thread for these events.

# VOICE_UNSET_EVENT

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_UNSET_EVENT,** | IOCTL low-level driver command |
| **&event_handle,** | pointer to an event HANDLE |
| **sizeof(HANDLE *),** | size of the event HANDLE |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

HANDLE event_handle;

**Purpose**
This command is used to disable signaling. The driver will no longer notify the calling function or application when a voice playback message is received from an XDS voice playback board.

**Returns**
The command will not return a value from the driver.

**Comments**
This command is used to disable the signaling feature of the driver for voice playback messages. Signaling may be re-enabled by issuing a VOICE_SET_EVENT command. This command should be issued before the driver is closed.

# VOICE_RESOURCE_RCV

| | |
|---|---|
| **BOOL DevIoControl(** | |
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_RESOURCE_RCV,** | IOCTL low-level driver command |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&vrData,** | pointer to VOICE_RESOURCE_RCV_DATA data structure |
| **sizeof(VOICE_RESOURCE_RCV_DATA),** | size of VOICE_RESOURCE_RCV_DATA data structure |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

| | |
|---|---|
| VOICE_RESOURCE_RCV_DATA | vrData; |

typedef struct voice_resource_rcv_data
{

| | |
|---|---|
| unsigned char board_number; | board number of an XDS voice resource board |
| unsigned char port_number; | voice resource channel |
| unsigned long data_length1; | size of data (in bytes) in user voice data buffer 1 (for the record size) |
| char msg[4]; | a char array used to pass messages from the driver to the application library (not to and from the board itself) |

}VOICE_RESOURCE_RCV_DATA, *PVOICE_RESOURCE_RCV_DATA;

**Purpose**
This command is used to receive driver-to-library application informational messages dealing with a voice resource board.

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

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_DATA_ERROR | no messages on voice resource message queue |

**Comments**

This command checks to see if there is any message on the voice resource driver-to-application information message queue. Messages returned from this command are used to tell the application information about the progress of voice playback or voice record on a voice resource board where an action is required by the application. An example would be a "P0" message to the application. This is a request from the driver for more data to be put (by the application) in the user buffer 1 for a specified voice resource channel. "P1" would be a request for more data in buffer 2 for a specified voice resource channel. A list of these messages is contained in the *Voice Resource Functions* chapter of the XDS Native Command Set Function Library For SCSA & H.100 Boards user manual (Amtelco P/N 251M024).

# VOICE_RESOURCE_XMT

**BOOL DevIoControl(**

| | |
|---|---|
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_RESOURCE_XMT,** | IOCTL low-level driver command |
| **&vrData,** | pointer to VOICE_RESOURCE_DATA data structure |
| **sizeof(VOICE_RESOURCE_DATA),** | size of VOICE_RESOURCE_DATA data structure |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

VOICE_RESOURCE_DATA vrData;

typedef struct voice_resource_data
{

| | |
|---|---|
| unsigned char board_number; | board number of an XDS voice resource board |
| unsigned char port_number; | voice resource channel |
| unsigned char Msg[4]; | a message array used to pass messages between the driver and application (not to and from the board itself) |
| void *pBuffer; | pointer to a buffer with data passed to the driver |
| unsigned long board_comd; | voice resource board DSP command |
| unsigned long data_length1; | size of data (in bytes) in user voice data buffer 1 |
| unsigned long data_length2; | size of data (in bytes) in user voice data buffer 2 |
| unsigned long buffer_length; | half the size of the user buffer (in bytes) |
| unsigned char num_buffer; | the number of the buffer to use in IOCTL call |

}VOICE_RESOURCE_DATA, *PVOICE_RESOURCE_DATA;

**Purpose**
This command is used to send a voice resource DSP command to a voice resource board.

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

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_DATA_ERROR | board not available |
| STATUS_NOT_IMPLEMENTED | an invalid XMT command was attempted to be sent |

**Comments**

This IOCTL command is used to send DSP commands directly to the voice resource module DSP.

A list of applicable commands and examples are listed in the *Voice Resource Board* chapter of the driver reference manual.

# VOICE_RESOURCE_WRITE_DPRAM_DATA

**BOOL DevIoControl(**
**hdriver,** XDS device driver handle
**(DWORD) VOICE_RESOURCE_WRITE_DPRAM_DATA,** low-level driver command
**&ram_info,** pointer to XDS_DPRAM data structure
**sizeof(XDS_DPRAM),** size of XDS_DPRAM data structure
**NULL,** (not used)
**0,** (not used)
**&data_length,** pointer to number of bytes returned
**NULL);** (not used)

XDS_DPRAM 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 a buffer with data to be written
} XDS_DPRAM, *PXDS_DPRAM

**Purpose**
This command is used to write information into the dual-ported RAM on the voice resource module of the XDS board specified in board_number.

**Returns**
The command 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**
This command is included in the driver to facilitate writing things such as a dial string to play on a voice resource channel. The driver will prevent writing to the last 16 bytes of the dual-ported RAM on the voice resource module. This area contains receive and transmit command queues.

# VOICE_RESOURCE_WRITE_DPRAM_PARAMETERS

**BOOL DevIoControl(**
**hdriver,** XDS device driver handle
**(DWORD) VOICE_RESOURCE_WRITE_DPRAM_PARAMETERS,** low-level driver command
**&pVparms,** pointer to VOICE_RESOURCE_PARAMETERS data structure
**sizeof(XDS_DPRAM),** size of VOICE_RESOURCE_PARAMETERS data structure
**NULL,** (not used)
**0,** (not used)
**&data_length,** pointer to number of bytes returned
**NULL);** (not used)

VOICE_RESOURCE_PARAMETERS pVparms;

typedef struct voice_resource_parameters
{
UCHAR board_number; the board number
ULONG offset; the offset in bytes into dual-ported RAM
void *buffer; pointer to a buffer with data to be written
unsigned char num_bytes; the size (in bytes) of the data to be written
}VOICE_RESOURCE_PARAMETERS, *PVOICE_RESOURCE_PARAMETERS;

**Purpose**
This command is used to write parameter information into the dual-ported RAM on the voice resource module of the XDS board specified in board_number.

**Returns**
The command 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**
This command is included in the driver to write 8 bit or 16 bit (num_bytes is either 1 for 8 bit or 2 for 16 bit) values to the voice resource module. The driver will prevent writing to the last 16 bytes of the dual-ported RAM on the voice resource module. This area contains receive and transmit command queues.

# VOICE_RESOURCE_READ_DPRAM

**BOOL DevIoControl(**
**hdriver,** XDS device driver handle
**(DWORD)VOICE_RESOURCE_READ_DPRAM,** low-level driver command
**&ram_info,** pointer to XDS_DPRAM data structure
**sizeof(XDS_DPRAM),** size of XDS_DPRAM data structure
**NULL,** (not used)
**0,** (not used)
**&data_length,** pointer to number of bytes returned
**NULL);** (not used)

XDS_DPRAM 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_DPRAM, *PXDS_DPRAM

**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 command 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 information on the voice resource DSP, such as configuration data.

# VOICE_RESOURCE_SET_EVENT

**BOOL DevIoControl(**

| | |
|---|---|
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_RESOURCE_SET_EVENT,** | IOCTL low-level driver command |
| **&event_handle,** | pointer to an event HANDLE |
| **sizeof(HANDLE *),** | size of the event HANDLE |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

HANDLE event_handle;

**Purpose**

This command is used to enable the signaling mechanism in the driver for voice resource boards. When enabled, the driver will notify the calling function or application when the driver has a driver-to-application information message for the application.

**Returns**

The command will return the following codes:

| | |
|---|---|
| STATUS_SUCCESS | success |
| STATUS_BUFFER_TOO_SMALL | size of data structure passed in is incorrect |
| STATUS_UNSUCCESSFUL | could not initialize event mechanism in driver |

**Comments**

To use the signaling mechanism, the application should create a user event handle and pass that handle value to the device driver using this IOCTL call. The application should then use some sort of wait mechanism, such as WaitForSingleObject() to monitor the receive thread for these events.

# VOICE_RESOURCE_UNSET_EVENT

**BOOL DevIoControl(**

| | |
|---|---|
| **hdriver,** | XDS device driver handle |
| **(DWORD)VOICE_RESOURCE_UNSET_EVENT,** | IOCTL low-level driver command |
| **&event_handle,** | pointer to an event HANDLE |
| **sizeof(HANDLE *),** | size of the event HANDLE |
| **NULL,** | (not used) |
| **0,** | (not used) |
| **&data_length,** | pointer to number of bytes returned |
| **NULL);** | (not used) |

HANDLE event_handle;

**Purpose**
This command is used to disable signaling. The driver will no longer notify the calling function or application when a voice playback message is received from an XDS voice playback board.

**Returns**
The command will not return a value from the driver.

**Comments**
This command is used to disable the signaling feature of the driver for voice resource information messages. Signaling may be re-enabled by issuing a VOICE_RESOURCE_SET_EVENT command. This command should be issued before the driver is closed.

This page was intentionally left blank.