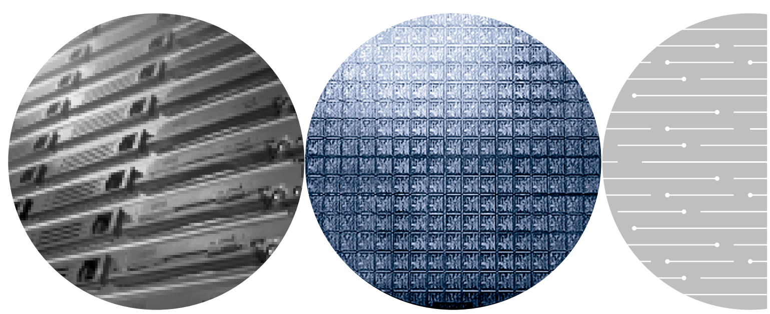
Application Note

Defining and Implementing Station Feature Sets

Intel in Communications



Contents

Executive Summary

Switching products from Intel that provide station interfaces and other types of resources are important building blocks in communications systems today. This document is designed to assist developers in defining and implementing station feature sets using these Intel® building blocks.

Introduction

Creating an application that uses Intel switching products to provide station interfaces and other types of resources in a communications system requires an understanding of station set features and an analysis of how different product features and behaviors impact the implementation of these features.

This document describes common station set features and then steps through an implementation of that feature set first for the modular station interface boards and then for Intel® Dialogic® Station Interface Boards and the Intel® NetStructure™ Station Interface Boards.

For a complete list of the function calls within the scope of the information in this document, see Appendix B.

Station Interface Boards from Intel

The information in this document applies to three types of boards:

Intel Dialogic Station Interface Boards

DISI16R2
DISI24R2
DISI32R2
DI0408LSAR2

Intel NetStructure Station Interface Boards

HDSI/480
HDSI/720
HDSI/960
HDSI/1200

Modular Station Interface Boards

MSI/80SC-GBL
MSI/80PCI-GBL
MSI/160SC-GBL
MSI/160PCI-GBL
MIS/240SC-GBL

The retirement of the modular station interface boards listed above was announced on February 11, 2003. The information in this document can guide developers in migrating from the retired boards to other station interface boards from Intel. See Appendix A for a product density matrix that can help determine the best replacement board based on resource requirements and a list of suggested replacement boards.

Abbreviations

For purposes of clarity in this document, the abbreviations in Table 1 will be used for board references.

Boards Referenced	Abbreviation
Intel Dialogic Station Interface Boards	DI
Intel NetStructure Station Interface Boards	HDSI
Modular Station Interface Boards	MSI

Table 1. Abbreviations for Board References

Common Station Set Features

Device Management

All station sets are managed using the ms_open() and ms_close() application programming interface (API) calls. Open and close operations are identical across all station set products.

The ms_open() API call is used to open a station interface device, and requires passing the name of the device as an ASCIIZ string, in the form msiB<b>C<c>, where <b> represents the board number, and <c> represents the station device number, which can range from 1 to the maximum number of station interfaces on the board.

The ms_open() call returns the device handle after the station device is successfully opened.

Device names can be found in the registry, under \\HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Configuration\Protocol Drivers. Devices with Spring Ware architecture are located in the DlgcSram folder and devices with DM3 architecture are located in the DLGCDM3 folder.

Closing a station device requires passing the device handle to the ms_close() API call.

CT Bus Routing

CT Bus routing is required to create and break audio connections between telephony resources. The CT Bus is synonymous to the SCbus, except that it is faster and supports 2048 timeslots (PCI version) instead of 1024.[1] In a system that contains both a CT Bus and an SCbus, the SCbus specifications are used by the system.

MSI boards do not include on-board voice resources. For this reason, MSI station sets are not supplied with a dedicated voice resource. Applications requiring voice resources must add boards with routable media resources to the system configuration and must use routable voice resources from those boards. The application using MSI station sets must provide a voice resource to a station set whenever it is required for voice operations.

Station sets using most DI and HDSI boards are provided with a permanently dedicated voice resource.[2] Because such a voice resource cannot be routed away from the station set to which it is dedicated, the voice resource assigned to a station set when the system service is starting cannot be changed.

To determine which voice resource is permanently routed to a particular station set, refer to the registry at \\HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Configuration\Protocol Drivers\DLGCDM3\Devices. All devices using the DM3 architecture are listed in this directory. Each station set is provided with a unique directory containing a set of key values for its use. The VoiceDevice key will contain the name of the voice resource that is dedicated to the station set.

All station sets can retrieve the CT Bus timeslot that they are transmitting on via the ms_getxmitslot() API call. In a static configuration using the DM3 architecture, the CT Bus timeslot that the voice resource transmits on must be obtained via ms_getxmitslot() instead of the voice API option dx_getxmitslot(). Calling dx_getxmitslot() on a voice resource from a static configuration will result in the "EDX_SH_MISSING" error message indicating that the CT Bus switching fabric is missing.

Station set audio connections are created and broken using calls to ms_listen() and ms_unlisten respectively. To create a full duplex audio connection between a station set with DM3 architecture and the voice resource in a fixed configuration, the SCbus/CTBus timeslot passed to the call to ms_listen() should be the SCbus/CTBus timeslot on which the station set is transmitting audio. In other words, the station set should "listen to itself." To create a full duplex audio connection between a station set and another telephony resource, the SCbus/CTBus timeslot passed to the call to ms_listen() is the return value of the xx_getxmitslot() API call.

[1]CT Bus with the CompactPCI* form factor supports 4096 timeslots.
[2]HDSI/1200 does not provide dedicated voice resources. DI0408LSAR2 provides media loads for either dedicated or routable voice resources.

Ring-Generation API Calls	MSI	HDSI	DI
ms_genring()	S	S	S
ms_genringex()	S	S	S
ms_genringCallerID()	U	S	S

Table 2. Supported and Unsupported API Calls

Signaling

All station sets retrieve the current hook status by passing the device handle to the ATMS_TSSGBIT() MSI API call. This function returns one of the two following values:

- MS_ONHOOK – station set is on-hook
- MS_OFFHOOK – station set is off-hook

Ring Generation and Caller ID (FSK) Signaling

The application can initiate generate-ring-cycles on a station set via the following MSI API function calls: ms_genring(), ms_genringEx(), and ms_genringCallerID(). MSI ring-generation capabilities are limited to ms_genring() and the extended ms_genringex() API call.

To provide frequency shift key (FSK) capabilities for CallerID, the HDSI and DI station sets support the ms_genringCallerID() API call, as well as the basic and extended MSI ring-generation functions. Table 2 lists the supported ring generation API calls for each of the station set product lines. Supported API calls are denoted with an “S” and unsupported calls with a “U”.

The basic ring generation API call is ms_genring(). It requires the device handle of the station set, the maximum number of generated ring cycles, and the API blocking mode (synchronous or asynchronous).

Distinctive ringing can be accomplished via the ms_genringex() and ms_genringCallerID() extended MSI API functions. Distinctive ringing requires that the application enable the MSG_DISTINCTRNG board level parameter via the ms_setbrdparm() MSI API call.

The call to ms_setbrdparm() requires a void pointer to a MS_CADENCE structure. The MS_CADENCE structure requires that the *cadid* field have a value between 1 and 8 to uniquely identify the cadence. The length of the cadence is specified in the *cadlength* field, and should be set to the default length of 6 seconds via MS_RNGA_CADLENGTH.

The final field of MS_CADENCE is a pointer to the cadence pattern. Table 3 lists the cadence patterns that are supported by different board types.

Note that MS_RNGA_SPLASH3 and MS_RNGA_SPLASH4 are not supported cadence patterns on the HDSI and DI boards. Using these patterns will not cause an error in ms_genringex() or ms_genringCallerID(), but no ring will be generated on the station set.

A distinctive ring will be assigned to the station and become the default ring cadence for that station. Future rings generated by either ms_genringex() using MS_RNG_DEFAULT or ms_genring() will use the new default ring cadence.

Caller identification can be transmitted while generating a ring on a station set with DM3 architecture via the call to ms_genringCallerID(), which allows the application developer to specify an FSK-formatted caller identification string. The ms_genringCallerID() function was added to relieve the application developer from the responsibility of setting up Caller ID functionality at the host level by implementing the FSK Caller ID transmission through the ms_genringCallerID() API function call. Note that using the MS_RNGA_SHORTLONG distinctive-ring cadence with ms_genringCallerID() will cause CallerID transmissions to fail. Table 4 lists the FSK group identifier tokens that are currently supported.

HDSI and DI boards have a *.config* file for each feature configuration description (FCD) file located in the <INSTALL DIRECTORY>\Dialogic\Data directory.

Cadence ID	MSI	HDSI	DI
MS_RNG_DEFAULT	S	S	S
MS_RNGA_TWOSEC	S	S	S
MS_RNGA_ONESEC	S	S	S
MS_RNGA_SPLASH1	S	S	S
MS_RNGA_SPLASH2	S	S	S
MS_RNGA_SPLASH3	S	U	U
MS_RNGA_SPLASH4	S	U	U
MS_RNGA_LONGSHORT	S	S	S
MS_RNGA_SHORTLONG	S	S	S

Table 3. Supported and Unsupported Cadence Patterns

The *.config* file can be used to alter default station behavior. The default ring cadence template can be modified by editing the Net_RingOn and Net_RingOff parameters located in the CAS section of the appropriate <country>_hdsi.config or di<product>.config file. The Net_RingOn and Net_RingOff parameters include the following attributes:

```
Pulse=<Signal ID>, <Off Pulse
Code>, <On Pulse Code>, <Pre Pulse
Interval>, <Min Pulse Interval>,
<Nominal Pulse Interval>, <Max
Pulse Interval>, <Post Pulse
Interval>
```

The following attributes can be modified, but all three parameters must have the same value: <Min Pulse Interval>, <Nominal Pulse Interval>, and <Max Pulse Interval>. Figure 1 illustrates how these attributes will affect the ring cadence template.

The ringing pattern in Figure 1 is used when the following Net_RingOn and Net_RingOff attributes are specified:

```
pulse=0xC15CA036,0xA4,0xAA,
0,2000,2000,2000,50    ! Net_RingOn

pulse=0xC15CA037,0xA4,0xA4,
50,3900,3900,3900,0    ! Net_RingOff
```

Whenever a change is made to a *.config* file, the *<INSTALL DIRECTORY>\Dialogic\Bin\fcdgen* utility must be executed with the location and name of the modified or new *.config* file. This utility will create a new FCD file, which must be specified in the *FCDFileName* field of the appropriate board listed in the configuration manager. See Figure 2 for an example.

FSK Group Identifier Tokens	Meaning
T:	Date and time
N:	Name
B:	Name absence reason
I:	Telephone number
A:	Telephone number absence reason
R:	User-defined data

Table 4. Meaning of FSK Group Identifier Tokens

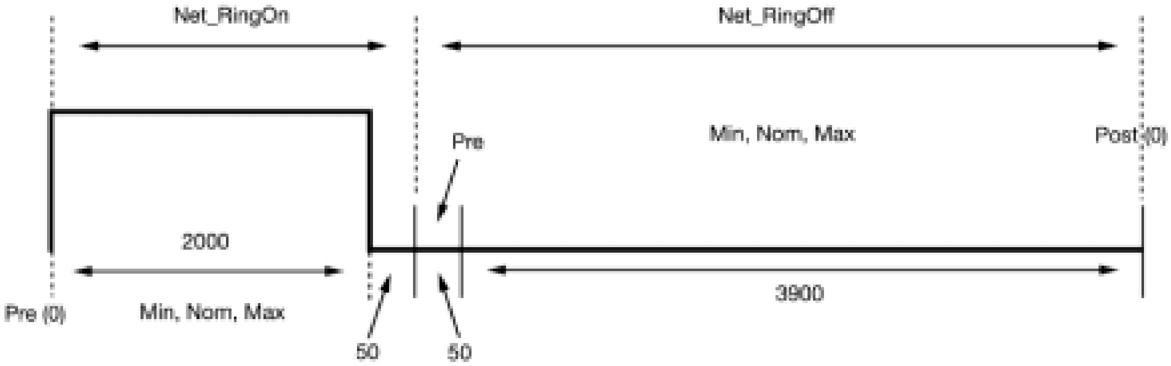


Figure 1. Ring Cadence Template Example

Message Waiting Indicator

The MSI API provides access to a message waiting indicator (MWI) via the call to ms_SetMsgWaitInd(), which generates an FSK signal that illuminates the message waiting LED. This API call requires that the application developer provide the device handle and one of the two state variables for the MWI state variable for MWI LED manipulation:

- MS_MSGINDON – turns MWI LED on
- MS_MSGINDOFF – turns MWI LED off

The ms_SetMsgWaitInd() is supported only on the HDSI and DI boards. This API function call relieves the application developer from the responsibility of implementing message waiting functionality via FSK messages at the host level.

Zip Tones

Zip tones are only supported on MSI station devices. Station sets with DM3 architecture can use their dedicated voice resources to generate a tone similar to a zip tone via dx_playtone() or dx_playtoneEx() instead of using the MSI API to generate an zip tone.

Conferencing

The DISI16R2, DISI24R2, and DISI32R2 boards have 16 conference resources with echo cancellation, and the DI0408LSAR2 has 9 conference resources with echo cancellation. MSI boards have 32 conference resources without echo cancellation. The HDSI products currently do not have conferencing resources.

Tone Generation and Detection

Because the MSI API does not provide function calls that generate or detect tone, the voice API should be used for tone generation and detection functionality. The application developer must follow the routability guidelines for various voice resource and station set combinations.

- For voice resource functionality on the Windows* operating system, refer to http://resource.intel.com/telecom/support/releases/winnt/SR511FP1/onldoc/htmlfiles/pgmgd3/1456-04.html
- For voice resource functionality on the Linux* operating system, refer to http://resource.intel.com/telecom/support/releases/unix51/linux51/SR5.1_Linux/Onldoc/html_files/vox_api/1453-02.html

To determine the tone generation and detection capabilities of a voice resource dynamically, use the dx_getfeaturelist() voice API call. The voice API header file *dxxxlib.h* defines each feature represented by a member in the FEATURE_TABLE data structure. The *ft_tone* field in the FEATURE_TABLE structure contains a bitmask that specifies the tone features supported on a particular voice device.

Play and Record

Because the MSI API does not provide function calls that play or record media, the voice API should be used for play and record functionality. The application developer must follow the routability guidelines for the various voice resource and station set combinations.

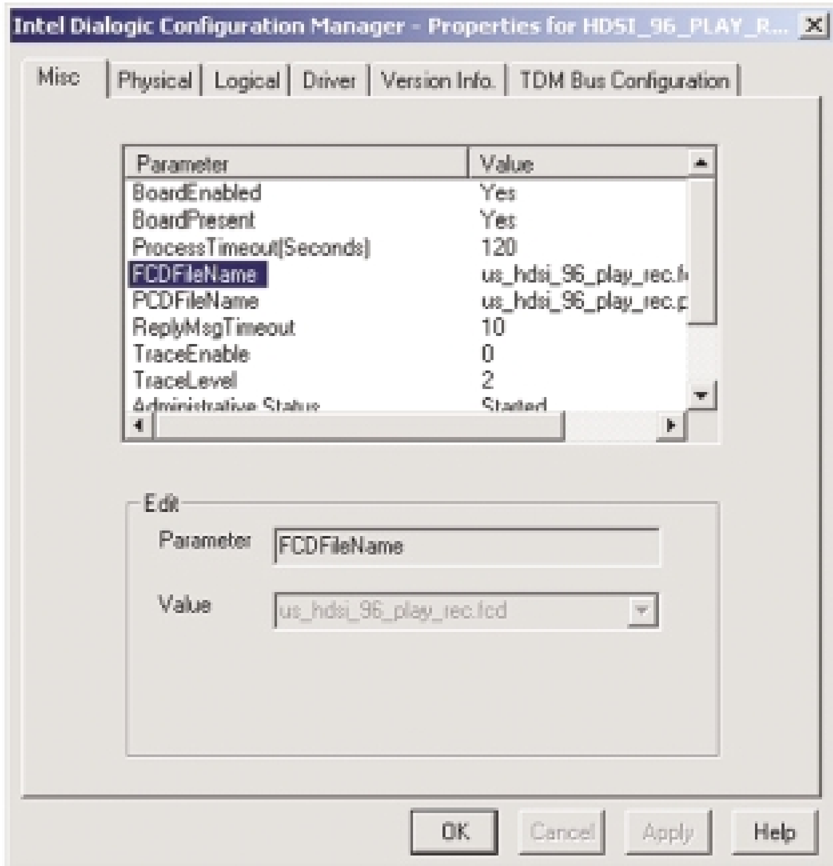


Figure 2. Configuration Manager Example

- For voice resource functionality on the Windows operating system, refer to http://resource.intel.com/telecom/support/releases/winnt/SR511FP1/onldoc/htmlfiles/pgmgd3/1456-04.html
- For voice resource functionality on the Linux operating system, refer to http://resource.intel.com/telecom/support/releases/unix51/linux51/SR5.1_Linux/Onldoc/html_files/vox_api/1453-02.html

To determine play and record functionality dynamically for all voice resources, use the dx_getfeaturelist() voice API call. The voice API header file *dxxxlib.h* defines each feature represented by a member in the FEATURE_TABLE data structure. The *ft_play* field contains a bitmask that specifies the play features supported on a particular voice device. The *ft_record* contains a bitmask that specifies the record features supported on a particular voice device.

Play Speed and Volume Control

HDSI boards do not support speed and volume control. DISI16R2, DISI24R2, and DISI32R2 do support speed and volume control.[3]

Play speed and volume can be controlled using voice API functions.

- To change the speed or volume explicitly, use dx_adjsv().

  The dx_adjsv() voice API call explicitly adjusts the speed or volume on the specified channel. The speed or volume can be set to a value, adjusted incrementally, or set to toggle. The speed and volume modification tables have 21 entries that represent different levels of speed or volume with ten levels above and below the default speed and volume. The dx_setsvmt() function calls the table with the explicit values that can be set.

- To modify speed or volume in response to specified conditions (i.e., if DTMF is “1”, increment speed by one level), use dx_setsvcond().

[3] For a list of other boards that support speed and volume control, see PTR 21973.

The dx_setsvcond() voice API call sets speed and volume adjustments and adjustment conditions for all subsequent play on the specified voice device. An adjustment condition can take place at the start of a play, or can be set for an incoming digit during a play. The adjustment condition can be changed or canceled at any time, and calls to dx_ setsvcond() are cumulative. Calling dx_clrsvcond() will clear the current speed and volume conditions.

API Data Structures

MSI Data Structures

MS_CADENCE

The MS_CADENCE data structure is used to specify the ring cadence pattern and length for the specified cadence ID.

```
typedef ms_cadence {
BYTE   cadid;        // Cadence ID, <1-8>
BYTE   cadlength;    // Cadence length
BYTE*  cadpattern;   // Pointer to cadence pattern
}  MS_CADENCE;
```

Voice Data Structures

FEATURE_TABLE

The FEATURE_TABLE data structure is used to specify the features that are supported on a particular device.

```
typedef struct feature_table {
   unsigned short ft_play;
   unsigned short ft_record;
   unsigned short ft_tone;
   unsigned short ft_e2p_brd_cfg;
   unsigned short ft_fax;
   unsigned short ft_front_end;
   unsigned short ft_misc;
   unsigned short ft_rfu[ 8] ;
}  FEATURE_TABLE;
```

Table 5 lists the features on the specified voice device that are specified in the bitmasks in the fields in the FEATURE_TABLE

Fields	Features
ft_play	Play
ft_record	Record
ft_tone	Tone
ft_e2p_brd_cfg	Board configuration
ft_fax	Fax
ft_frontend	Front end
ft_misc	Miscellaneous

Table 5. Features Specified in FEATURE_TABLE Fields

DX_SVCB

The speed/volume condition block (DX_SVCB) data structure is used to specify a play adjustment condition that can adjust play speed or volume at the beginning of playback or in response to digits entered by the user during playback.

```
typedef struct dx_svcb {
   unsigned short type;    // Bitmask
   short adjsize;          // Size of adjustment
   unsigned char digit;    // Digit causing action
   unsigned char digtype;  // DTMF = 0
}  DX_SVCB;
```

DX_SVMT

The speed/volume modification table (DX_SVMT) data structure has 21 entries that represent different levels of speed or volume, and is used to specify the rate of change for speed or volume adjustments by channel. Values are set or retrieved from the table with dx_setsvmt() and dx_getsvmt().

```
typedef struct dx_svmt {
   char decrease[  10 ];          // 10 downward steps
   char origin;                   // Origin speed and volume
   char increase[  10 ];          // 10 upward steps
}  DX_SVMT;
```

MSI API Support in Other Intel® Products

HDSI

The following calls are not supported because HDSI boards do not currently have conferencing resources or capabilities:

ms_addtoconf
ms_chgxtder
ms_delconf
ms_delxtdcon
ms_estconf
ms_estxtdcon
ms_getcnflist
ms_monconf
ms_remfromconf
ms_setcde

Zip tones are used when the station set user has a headset instead of a telephone. Since the headset cannot ring, a zip tone is generated. Since HDSI boards have on-board voice resources, zip tones are not necessary and ms_genziptone is not supported because the voice resource can generate a tone.

DI

The following calls are not supported because conference connection extender functions are not available in DI boards:

ms_chgxtder
ms_delxtdcon
ms_estxtdcon

Zip tones are used when the station set user has a headset instead of a telephone. Since the headset cannot ring, a zip tone is generated. Since HDSI boards have on-board voice resources, zip tones are not necessary and ms_genziptone is not supported because the voice resource can generate a tone.

MSI

The following calls are not supported. Additional information is provided below each call.

ms_genringCallerID

The ms_genringCallerID() call was added to relieve the application developer from implementing FSK Caller ID at the application level. DI and HDSI boards have FSK capabilities on their on-board voice resources, and can handle the FSK generation internally.

ms_ResultMsg and ms_Result Value

The ms_ResultMsg and ms_ResultValue calls were added for the enhancement of station set error messages.

ms_SendData

The ms_SendData() call was added to send FSK data to a station that is off-hook. The DI and HDSI boards have FSK capabilities on their on-board voice resources, and can handle the FSK generation internally.

ms_SetMsgWaitInd

The ms_SetMsgWaitInd call uses FSK data to turn the message waiting light on or off. The DI and HDSI boards have FSK capabilities on their on-board voice resources, and can handle the FSK generation internally.

Appendix A. Product Density Matrix

The product density matrix in Table 6 is designed to help determine the best hardware solution based on basic resource requirements. In the media load column, *xx* represents the country code for the particular media load.

Refer to http://www.intel.com/design/network/products/telecom/boards/switching.htm for another comparison of product features with links to product datasheets.

Board	Media Load	Station Intrfcs	Conference Rsrcs	Voice Dvcs	Network Intrfcs	Fax Rsrcs	CSP[4]
MSI/80SC-GBL	N/A	8	32	0	0	0	0
MSI/80PCI-GBL	N/A	8	32	0	0	0	0
MSI/160SC-GBL	N/A	16	32	0	0	0	0
MSI/160PCI-GBL	N/A	16	32	0	0	0	0
MSI/240SC-GBL	N/A	24	32	0	0	0	0
HDSI/480	xx_hdsi_48_play_rec	48	0	48	0	0	0
HDSI/720	xx_hdsi_72_play_rec	72	0	72	0	0	0
HDSI/960	xx_hdsi_96_play_rec	96	0	96	0	0	0
HDSI/1200	xx_hdsi	120	0	0	0	0	0
DISI16R2	disi16	16	16	16	0	0	0
DISI24R2	disi24	24	16	24	0	0	0
DISI32R2	disi32	32	16	32	0	0	0
DI0408LSAR2	di0408lsa – Media Load 1	8	9	12	4	2	0
DI0408LSAR2	di0408lsa – Media Load 2	8	9	8[5]	4	2	0
DI0408LSAR2	di0408lsa – Media Load 4	8	9	8[6]	4	2	4

Table 6. Product Density Matrix

The following table lists the retired boards discussed in this document and suggested replacement boards.

Retired Board	Suggested Replacement Boards
MSI/80SC-GBL	DI0408LSAR2
MSI/80PCI-GBL	DI0408LSAR2
MSI/160SC-GBL	DISI16R2
MSI/160PCI-GBL	DISI16R2
MSI/240SC-GBL	DISI24R2, DISI32R2, HDSI[xxxPCI][7]

Table 7. Recommended Replacements for Retired Boards

[4]Continuous speech processing
[5]Routable voice
[6]Routable voice
[7]Any HDSI board with the PCI form factor

Appendix B. Function Call List

The following function calls are included within the scope of the information in this document.

MSI API	Voice API
ATMS_TSSGBIT	dx_adjsv
ms_close	dx_clrsvcond
ms_genring	dx_getfeaturelist
ms_genringex	dx_getxmitslot
ms_genringCallerID	dx_listen
ms_getxmitslot	dx_playtone
ms_listen	dx_playtoneEx
ms_open	dx_setsvcond
ms_setbrdparm	dx_setsvmt
ms_SetMsgWaitInd	dx_unlisten
ms_unlisten	

Appendix C. Known Issues

The following problem tracking records (PTRs) describe known issues in respect to the boards discussed in this paper.

- PTR 22914 – ATMS_TSSGBIT() will return MS_ONHOOK if the station set is off-hook when ms_open() is called
- PTR 22919 – dx_playtone() can only play up to 35 unique tones on DI boards
- PTR 26866 – If ms_close() is called while the station set is off-hook, the system service must be restarted for proper operation of the station set
- PTR 28203 – Intel Dialogic DI/0408-LS-A does not detect a remote phone going off-hook in an edge condition. The board never sends up a connected event even though a call is connected.
- PTR 29235 - DX_MAXNOSIL termination condition does not accurately terminate when expected (i.e., setting a maximum non-silence termination condition of two seconds may cause actual termination after more than two seconds of non-silence)

To learn more, visit our site on the World Wide Web at **http://www.intel.com**.

1515 Route Ten
Parsippany, NJ 07054
Phone: 1-973-993-3000