Dialogic
Making Innovation Thrive™

Dialogic® NaturalAccess™ ISDN Management API Developer's Manual

October 2009 64-0511-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
9000-6722-10 Beta	September, 1999	NBS, for CT Access 3.0
9000-6722-11	July, 2000	EPS / SJC, Support for Fusion 4.0
9000-6722-12	September, 2000	SJC, for CT Access 3.0 or 4.0
9000-6722-13	March, 2001	SJC, for NACD 2000-2
9000-6722-14	April, 2001	SJC, for NACD 2001-1 Beta
9000-6722-15	August, 2001	SJC, for NACD 2001-1
9000-6722-16	November, 2001	SJC, for NACD 2002-1 Beta
9000-6722-17	May, 2002	LBG, NACD 2002-1
9000-6722-18	November, 2002	LBG, NA 2003-1 Beta
9000-6722-19	April, 2003	MVH, NA 2003-1
9000-6722-20	April, 2004	SRR, NA 2004-1
64-0511-01	October, 2009	LBG, NaturalAccess R9.0
Last modified: September 4, 2009		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

The *Dialogic® NaturalAccess™ ISDN Management API Developer's Manual* describes how to use the NaturalAccess ISDN Management (IMGT) API to interact with the ISDN stack and provide access to information and messages that are not part of traditional call control.

This manual is for developers of ISDN applications. The developer should be familiar with NaturalAccess, basic telephony concepts, switching, and the C programming language.

2 Terminology

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

Former terminology	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries

Former terminology	Dialogic terminology
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel

3 NMS ISDN Management API overview

Integrated Services Digital Network (ISDN)

Integrated Services Digital Network (ISDN) is a continually evolving international standard for networking services, including voice and non-voice services. The network is completely digital, from one end to the other. Voice information is digitized and sent in digital form. Signaling information is sent separately using a method called common channel signaling (CCS).

The NMS ISDN Management (IMGT) service is a Natural Access service that interacts with the NMS ISDN protocol stack and provides access to information and messages that are not part of traditional call control. The IMGT service provides functions that enable you to put B channels in and out of service and query the stack about the status of a particular B channel or D channel. The service also reports some variant-specific messages.

IMGT applications run independently from call control applications, whether the call control application is based on the ISDN Messaging API or the Natural Call Control (NCC) API.

ISDN carriers

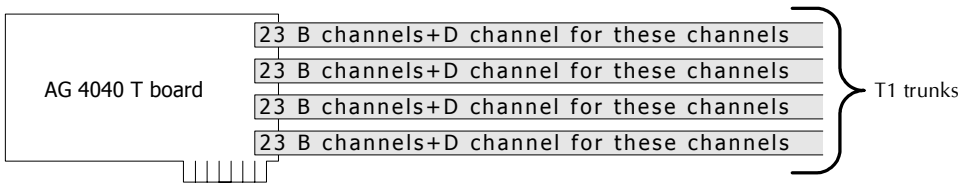
ISDN is transmitted over standard T1 and E1 carriers. T1 and E1 carriers are typically four-wire digital transmission links. T1 is used mainly in the United States, Canada, Hong Kong, Taiwan, and Japan. E1 is used in most of the rest of the world.

Data on a T1 or E1 trunk is transmitted in channels. Each channel carries information digitized at 64000 bits per second (bps).

Primary-rate ISDN channels

For primary rate ISDN, T1 carries 24 channels. E1 carries 32 channels. The channels are usually used as follows:

- On a T1 trunk, 23 of the 24 channels carry data: voice, audio, data and/or video signals. These channels are called bearer channels (B channels).
- On an E1 trunk, 30 of the 32 channels are B channels.
- On a T1 or E1 trunk, one channel carries signaling information for all B channels. This is called the D channel.



T1 trunk (standard configuration)

NFAS channel usage

In setups with multiple T1 ISDN trunks, a non-facility associated signaling (NFAS) configuration is sometimes used. In this configuration, the D channel on one of the ISDN trunks carries signaling for all channels on several other trunks. This configuration leaves channel 24 free on each of the other trunks to be used as another B channel.

Sample NFAS configuration

NFAS configurations are supported only for T1 trunks.

NMS ISDN maintenance messages

NMS ISDN maintenance messages are layer 3 ISDN messages used by ISDN to put B channels and D channels in the in-service or out-of-service state. These messages are supported only on North American variants: 4ESS, 5ESS, NI2, and DMS-100. For 5ESS only, service messages can also be used to place the entire interface in or out of service.

There are two types of IMGT service maintenance messages:

- SERVICE messages
- RESTART messages

SERVICE messages

In some ISDN variants, SERVICE messages can be exchanged to put channels in and out of service. Through the IMGT service, the application can send SERVICE messages to the trunk and receive SERVICE or SERVICE ACKNOWLEDGE events. When a SERVICE message is received, the ISDN stack automatically sends a SERVICE ACKNOWLEDGE. No primitives are needed to send a SERVICE ACKNOWLEDGE. The application can also query the stack to learn the status of a particular B channel.

SERVICE message handling is implemented for the following variants:

- DMS
- AT&T 4ESS
- AT&T 5ESS
- NI2

RESTART messages

Using the IMGT service, the application can receive notification whenever a RESTART message is received. When a RESTART message is received, the ISDN stack automatically sends a RESTART ACKNOWLEDGE.

For the DMS variant, the reception of a RESTART message resets the status of the B channel to the in-service setting.

IMGT service software components

The NMS ISDN software package includes the following IMGT service software components:

- A *readme* file
- The NMS ISDN Management API function library
- Header files
- Run modules
- Board keyword files
- A demonstration program with the source code file and *makefile*

Readme file

The ASCII text file *readme_isdn.txt* contains release information that does not appear in other documentation. Consult this file to learn where the NMS ISDN software components are located after installation.

NMS ISDN Management API function library

The IMGT service runs on the host computer. The library is used by the application program to interact with the ISDN protocol stacks running on the board in bringing B channels in and out of service in certain ISDN variants.

Note: DPNSS is not supported by the IMGT service.

The IMGT service is supplied as an extension to the ADI service. The IMGT library is a dynamic-link library (DLL) under Windows and a shared object under UNIX. The libraries have different names under different operating systems.

Operating system	Natural Access library name
Windows	*imgtapi.lib*, *imgtapi.dll*
UNIX	*libimgtapi.so*

Header files

Two header files for the IMGT service are supplied with NMS ISDN software:

File name	Description
imgtdef.h	Contains function prototypes, event codes, and associated data structures.
imgtsvc.h	Contains primitive codes and associated data structures.

Run modules

The run module contains the basic low-level software that a board requires to support the IMGT service. The module is transferred from the host into on-board memory when the board boots.

Different run modules are supplied for different configurations and are specific to the protocol variant and country. The module you use depends upon what board type you are using. For more information about run modules, see the *NMS ISDN Installation Manual*.

Board keyword files

Board keyword files contain information that determines how to set up your boards for use. These files also contain country-specific information and define what trunks are assigned to which D channels.

Several example files are included, describing ISDN configurations for different boards. Use these files to create a file describing your hardware and software setup. For details, see the *NMS OAM System User's Manual*.

Demonstration program

A demonstration program (*imgtdemo*) is included as an executable program with its source code file and *makefile*. Use this demonstration program to start the IMGT service. You can then monitor B channel status and put it in and out of service.

Other components

In addition to the NMS ISDN software, you need the following components to build an NMS ISDN protocol application:

- One or more NMS T1 or E1 trunk interface boards
- Natural Access

Warning: 	NMS Communications obtains board-level approvals certificates for supported countries. Some countries require that you obtain system-level approvals before connecting a system to the public network. To learn what approvals you require, contact the appropriate regulatory authority in the target country.

Natural Access

Natural Access is a complete development environment for telephony applications. It provides a standard set of telephony functions grouped into logical services. Natural Access services provide functions for telephony-related tasks such as call control, tone generation and detection, and voice playing and recording.

The Natural Access Switching service controls switching on MVIP-compliant devices. You can use this service to make or break connections, send patterns, and sample data, among other tasks. This service supports both MVIP-95 and MVIP-90 specifications. Alternatively, you can use the *swish* utility to control switching interactively or in a batch mode.

For information about installing and using Natural Access, see the Natural Access documentation.

Contexts and event queues

Natural Access organizes services and accompanying resources around a single processing unit called a context. To access service functionality, an application creates a context and attaches the services it requires. A context usually represents an application thread performing a related set of functions, such as controlling a single telephone call.

Natural Access provides multi-processing support. Multiple Natural Access application processes can perform tasks on behalf of the same context (referred to as context sharing). Natural Access applications can transfer control of contexts (for example, contexts associated with individual telephone calls) to other Natural Access applications (referred to as context hand-off).

An event queue is the communication path from a Natural Access service to an application. A Natural Access service generates events indicating certain conditions or state changes and sends them to applications through the event queue.

The Natural Access Server (*ctdaemon*) must be running for NMS OAM to be available. If *ctdaemon* is stopped, all dependent applications receive an error.

For detailed information about Natural Access, see the *Natural Access Developer's Reference Manual*.

oamsys and the OAM configuration file

When you set up the system, specify a board keyword file for each board. This file specifies whether or not a board performs MVIP switching, which board is the MVIP clock master, which software modules to transfer to the board's memory on startup (including which TCPs to load), and other settings. An OAM system configuration file references these board keyword files.

Run *oamsys* to configure your boards based on the information in the OAM system configuration file and the board keyword files it references. *oamsys* transfers all software modules specified in the file to each board and performs any other configuration activities specified in the board keyword files. You can also start *oammon* to monitor the boards for errors and other events.

Use *oamcfg* to change system information or board parameters after the system is running. *oamsys*, *oamcfg*, and *oammon* are installed with Natural Access and require Natural Access to run in Server mode.

To learn how to modify NMS OAM board keyword and system configuration files to set up your NMS ISDN software, see the *NMS ISDN Installation Manual* and the *NMS OAM System User's Manual*

Developing an IMGT application

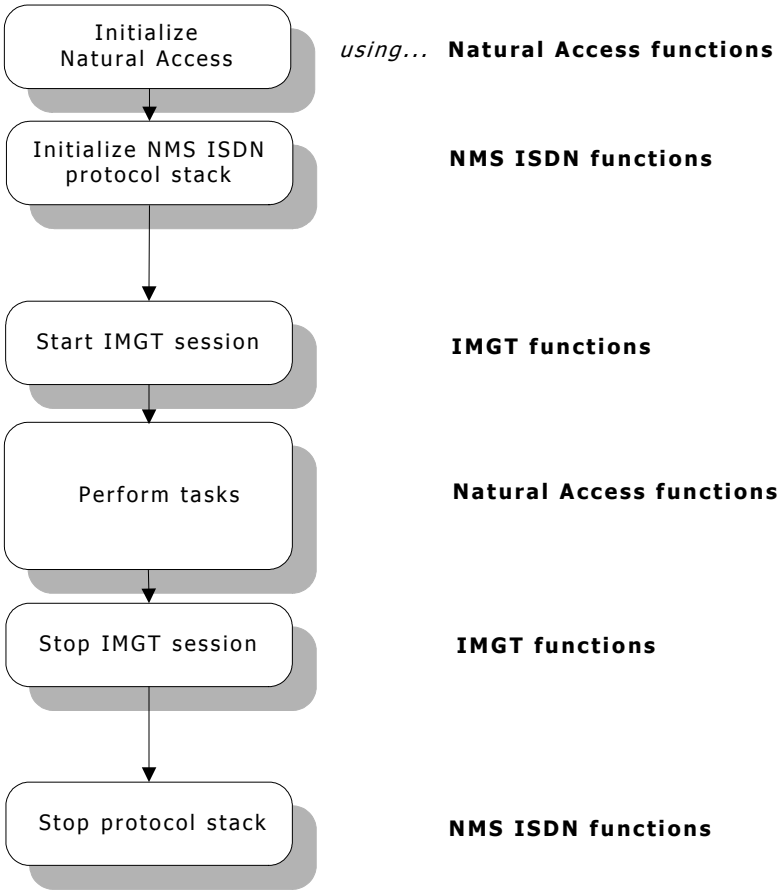
Perform the following steps to create an IMGT application:

Step	Action
1	Install digital trunk interface boards in a system, and any other boards the application requires. Refer to the board installation manuals.
2	Install Natural Access. Refer to the *Natural Access Installation* booklet.
3	Install the NMS ISDN software for each country or region in which your application will be used. Refer to the *NMS ISDN Installation Manual.*
4	Edit the system configuration file and associated board keyword files so that they provide configuration information for all boards in your system. Refer to the *NMS ISDN Installation Manual,* installation manuals for your boards, and the *NMS OAM System User's Manual.*
5	Test the hardware installation. Refer to the installation manuals for your boards.
6	Write the management application. Refer to this manual and the Natural Access documentation set.

4 Using the IMGT service functions

IMGT application overview

An IMGT application typically performs the following tasks:



IMGT application flowchart

The following table summarizes the illustrated tasks:

In this phase...	The application...
Initialize Natural Access	Initializes Natural Access services and creates one context for each B channel and D channel.
Initialize NMS ISDN protocol stack	Calls **isdnStartProtocol** to start up an ISDN protocol stack instance on each D channel context.
Start IMGT session	Invokes **imgtStart** to start an IMGT session on each D channel context.
Perform tasks	Uses the standard Natural Access event structure to send IMGT messages and receive IMGT events.

In this phase...	The application...
Stop IMGT session	Invokes **imgtStop** to end the IMGT session.
Stop protocol stack	Calls **isdnStopProtocol** to stop the ISDN protocol stack instance on each D channel context.

Setting up the Natural Access environment

Application setup for Natural Access consists of the following steps:

Step	Action
1	Initialize Natural Access for the process.
2	Create event queues.
3	Create contexts for each event queue.
4	Open services on each context.

Services are opened on a context by calling **ctaOpenServices** and passing a context handle and a list of service descriptors. Each service descriptor specifies the name of the service, service manager, and service-specific arguments.

The call to **ctaOpenServices** is asynchronous and returns immediately. When all services have been opened, the CTAEVN_OPENSERVICES_DONE event is returned to the application. Natural Access supports opening and closing services on an as-needed basis to share resources.

There can be only one open instance of each service per context. Once the IMGT service is opened, the application can make function calls from the IMGT service.

For details on setting up the Natural Access environment, see the *Natural Access Developer's Reference Manual.*

Setting up the IMGT service

Before an application can use functions from the IMGT service, the application must initialize and open the IMGT service on an open context.

Note: Remember to start the NMS ISDN protocol stack before starting the IMGT service.

Initializing the IMGT service

To open the IMGT service, include the IMGT service and the ADI Service Manager in the call to **ctaInitialize**.

The following code excerpt demonstrates initializing the IMGT service together with the ADI service:

```
void MyServiceInit()
{
    DWORD ret;
    CTA_INIT_PARMS initparms = { 0 };
    CTA_ERROR_HANDLER hdlr;
    CTA_SERVICE_NAME InitServices[] =              /* For ctaInitialize */
    {   { "ADI",  "ADIMGR" },
        { "IMGT", "ADIMGR" },
    };
                              /* Initialize size of init parms structure */
    initparms.size = sizeof(CTA_INIT_PARMS);

    if ( ( ret = ctaInitialize(InitServices,
                              sizeof(InitServices)/sizeof(InitServices[0]),
                               &initparms)) != SUCCESS)
    {
                                    /* Handle error conditions here.... */
    }
}
```

After invoking **ctaInitialize**, the application must:

1. Create an event queue attached to the ADI Service Manager by calling **ctaCreateQueue**. ADIMGR can be explicitly specified in the call. If NULL is passed, all service managers specified in **ctaInitialize** are attached.
2. Create a context by calling **ctaCreateContext**.

Opening the IMGT service

The IMGT service must be opened on a context to use the IMGT service's library of C functions. Opening the IMGT service starts the trunk monitor software on the board and enables it to make status information accessible to the application at specific intervals. Only one instance of the IMGT service can be opened per context.

When the application opens a service, it specifies a board. When the application opens the trunk monitor, it binds it to a context.

To open the IMGT service on a context, the application invokes **ctaOpenServices**. This function takes an array of CTA_SERVICE_DESC structures as an input argument. Each CTA_SERVICE_DESC structure defines a service (in this case, the IMGT service). If the service is opened successfully, a CTAEVN_OPEN_SERVICES_DONE event with the reason CTA_REASON_FINISHED is delivered to the application.

The CTA_SERVICE_DESC structure is defined as follows:

```
typedef struct CTA_SERVICE_DESC
{
    CTA_SERVICE_NAME name;        /* service name                          */
    CTA_SERVICE_ADDR svcaddr;     /* reserved                              */
    CTA_SERVICE_ARGS svcargs;     /* passes service-specific arguments     */
    CTA_MVIP_ADDR mvipaddr;       /* board #, stream, timeslot, mode       */

}CTA_SERVICE_DESC;
```

The following code sample demonstrates opening the IMGT service:

```
DWORD ret ;
CTA_EVENT event ;
CTA_SERVICE_DESC services[] =
{
   { {"IMGT", "ADIMGR"}, { 0 }, { 0 }, { 0 } } ,
} ;

                                              /* Open the IMGT service */
ret = ctaOpenServices(  ctahd,                /* A context handle      */
                        services,
                        sizeof(InitServices)/sizeof(InitServices[0]),);
if(ret != SUCCESS)
{
                                        /* Opening IMGT service failed */
    printf ( "OpenServices failed\n" );
}
else
{
                     /* Wait for the CTAEVN_OPEN_SERVICES_DONE event */
     while (1)
     {
       ctaWaitEvent( ctaqueuehd, &event, CTA_WAIT_FOREVER);

       if (event.id == CTAEVN_OPEN_SERVICES_DONE)
       {
                                    /* Check the reason of completion */
          if (event.value != CTA_REASON_FINISHED)
          {
            printf ( "Open services failed\n" );
          }

          break;
       }
       else
       {
                               /* Process other (unrelated) messages */
          ...
       }
     }
}
```

Starting a management session

Call **imgtStart** to start an IMGT session. The function takes as arguments the network access identifier (NAI) and the pointer to a configuration structure (IMGT_CONFIG structure). The configuration structure enables applications to filter any IMGT events that the application receives. After invoking **imgtStart**, the application can use **imgtSendMessage** to send messages to the IMGT service.

Network access identifiers (NAIs)

A trunk is referred to by its NAI. To initialize an ISDN protocol stack instance for a context (using **isdnStartProtocol**), specify the NAI of the trunk to associate with the context. From then on, the application can communicate with the D channel on that trunk by specifying the associated context handle. For example, when an event is received, the context handle indicates the trunk on which the event occurred. For more information about using **isdnStartProtocol**, see the *NMS ISDN Messaging API Developer's Reference Manual*.

Different board types support different numbers of D channels with different corresponding NAI numbers. The following table shows what each board type supports:

Digital trunk interface board type	Number of D channels	Default NAI values	Number of NFAS groups
Four-trunk boards	1 to 4	0 to 3	1 to 4
Two-trunk boards	1 or 2	0 or 1	1 or 2

If your configuration involves NFAS groups, the NAI for each group is specified in the system configuration file. For more information, see the *NMS ISDN Installation Manual*.

Sending IMGT messages to the stack

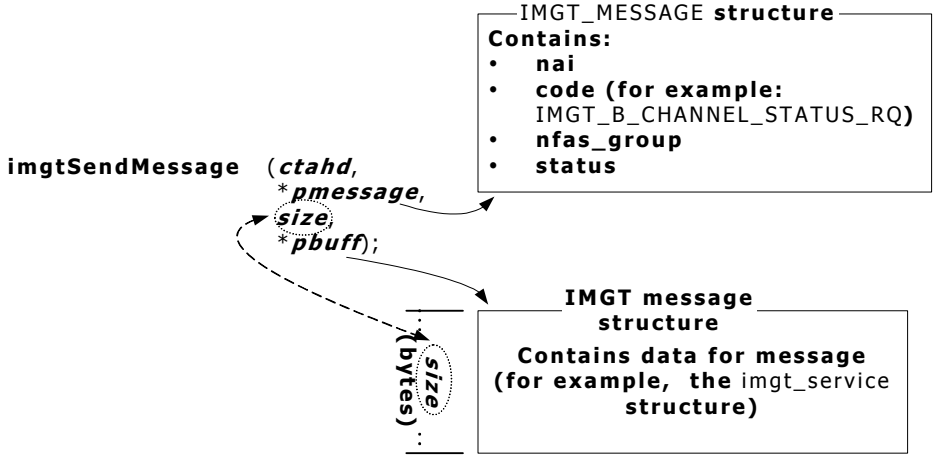
To send a message to the NMS ISDN protocol stack, the application builds two structures:

Structure	Description
IMGT_MESSAGE	In this structure, the application specifies the message to be sent by using one of the message primitives documented in IMGT data structures. The message primitive appears in the code field in this structure. In an NFAS configuration, the NAI in the IMGT_MESSAGE structure must be the same as the NAI carrying the D channel.
A primitive-specific data structure (if needed)	For messages that require additional data, a message structure is sent containing the data. The data differs for each message type. For details about seach message type, see *IMGT data structures* on page 35.

In addition, the application calls **imgtSendMessage** and specifies the following arguments:

Argument	Description
ctahd	The context identifier associated with the trunk on which the call is taking place.
pmessage	A pointer to IMGT_MESSAGE.
size	The size of the message structure.
pbuff	A pointer to the primitive-specific data structure.

The following illustration shows the content and meaning of each of the arguments sent in **imgtSendMessage**:



imgtSendMessage data structures

IMGT_MESSAGE is defined as follows:

```
typedef struct IMGT_MESSAGE
{
  BYTE nai;          /* Network access interface index           */
  BYTE code;         /* Primitive code                           */
  WORD nfas_group;   /* NFAS group number, only required for     */
                     /* configurations with duplicate NAI values */
  DWORD status;      /* Status or error for START/STOP           */
} IMGT MESSAGE;
```

Note: When NFAS is in use, the application should set ***nai*** in the IMGT_MESSAGE structure to the NAI that is carrying the D channel.

Receiving events and IMGT messages

The IMGT service sends messages to the application by means of a standard Natural Access event structure. The information arrives as a buffer attached to an IMGTEVN_RCV_MESSAGE event. See *IMGT data structures* on page 35 for information about the contents of the buffer.

This topic describes:

- Receiving events
- Receiving IMGT messages
- Stopping a management session

Receiving events

The events returned can be standard Natural Access events, events sent by an ISDN protocol stack instance or IMGT session, or events specific to any Natural Access extensions. They arrive in the form of the standard event data structure:

```
typedef struct CTA_EVENT
{
    DWORD    id;              /* Event code (and source service id)     */
    CTAHD    ctahd;           /* Context handle                         */
    DWORD    timestamp;       /* Timestamp                              */
    DWORD    userid;          /* User id (defined by ctaCreateContext)  */
    DWORD    size;            /* Size of buffer if buffer != NULL       */
    void     *buffer;         /* Buffer pointer                         */
    DWORD    value;           /* Event status or event-specific data    */
    DWORD    objHd;           /* Service client-side object handle      */
} CTA_EVENT;
```

The CTA_EVENT structure informs the application about events that occur on particular contexts, and provides additional information specific to the event. The event's prefix indicates the NMS service with which the event is associated. A partial list of these prefixes is shown in the following table:

This prefix...	Indicates...
CTAEVN	A Natural Access event
NCCEVN	An NCC service event
ADIEVN	An ADI service event
ISDNEVN	An NMS ISDN event
IMGTEVN	An IMGT service event

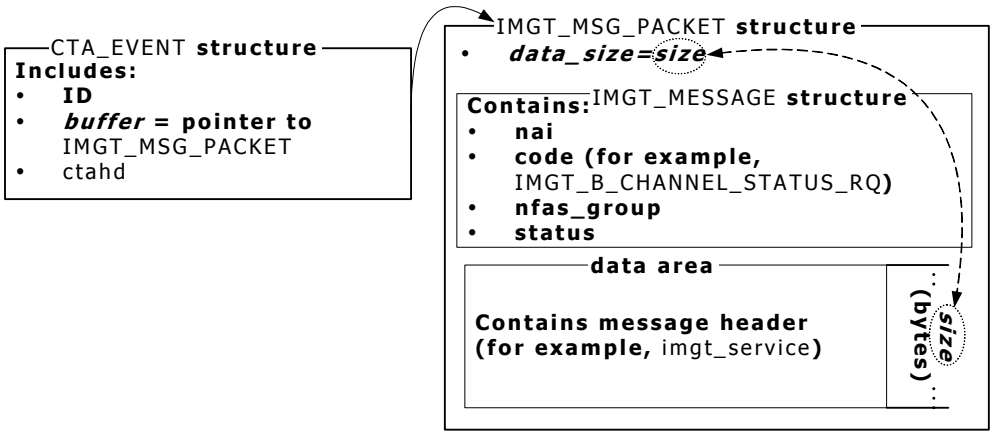
To receive these events, the application can invoke **ctaWaitEvent**.

Receiving IMGT messages

When an IMGT message is received, an IMGTEVN_RCV_MESSAGE event occurs. In the Natural Access structure, ***buffer*** is a pointer to an IMGT_MSG_PACKET structure. This structure contains:

- An IMGT_MESSAGE structure that contains the message and other data.
- A data area containing the message header.

The following illustration shows the structure of this message packet:



Receiving IMGT messages

Most IMGT events have a buffer associated with them. When the ***buffer*** and ***size*** fields of the Natural Access event indicate that a buffer is associated with the event, the application processes the buffer and must release it as soon as possible by calling **imgtReleaseBuffer**. Otherwise, the IMGT service interface runs out of buffers and stops passing events to the application.

If the configuration includes NAIs whose values are not unique on a given board, the ***nfas_group*** field of the IMGT_MESSAGE structure contains the NFAS group number for the NAI.

Stopping a management session

Use **imgtStop** to end an IMGT session.

5 Function reference

Function summary

Function	Synchronous/ Asynchronous	Description
imgtReleaseBuffer	Synchronous	Releases a buffer after the application is done processing it.
imgtSendMessage	Asynchronous	Sends a message to the management manager with attached data.
imgtStart	Asynchronous	Starts the IMGT service on a board.
imgtStop	Asynchronous	Stops a management session initiated with **imgtStart**.

Using the function reference

This section provides an alphabetical reference to the NMS ISDN Management service functions. A prototype of each function is shown with the function description and details of all arguments and return values. A typical function description includes:

Prototype	The prototype is shown followed by a listing of the function's arguments. Data types include: • WORD (16 bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is shown. Refer to *IMGT data structures* on page 35 for a description of all data structures and parameters.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to *NMS ISDN Management service errors* on page 59 for a list of all errors returned by NMS ISDN Management API functions.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Refer to *NMS ISDN Management service events* on page 61 for details about NMS ISDN Management API events and reason codes.
Example	Example functions that start with `Demo` are samples taken from demonstration function libraries shipped with the product. Example functions that start with `my` are excerpts taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

imgtReleaseBuffer

Releases a buffer after the application has completed processing the message.

Prototype

DWORD **imgtReleaseBuffer** (CTAHD ***ctahd***, void ****buffer***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
buffer	Pointer to buffer to be released.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The specified context handle is invalid.
IMGTERR_INVALID_BUFFER	***buffer*** does not point to a valid buffer.

Events

None.

Details

This function sends an indication to the IMGT manager that the application has finished processing an event buffer (described by the CTA_EVENT ***buffer*** and ***size*** fields) and is returning that buffer to the IMGT service.

The application must return every event buffer to the IMGT service as soon as possible or the IMGT service can run out of buffers and stop passing events to the application.

Example

```
void MyEventHandler( CTAHD ctahd )
{
   DWORD ret;
   CTA_EVENT event;
   char *errortext="";

   while( 1 )
   {
       ret = ctaWaitEvent( ctahd, &event, 100 );

       switch( event.id )
       {
    case CTAEVN_WAIT_TIMEOUT:
        break;
       case IMGTEVN_RCV_MESSAGE:
          /* process the buffer */
          ...
          ret = imgtReleaseBuffer( ctahd, event.buffer );
          if (ret != SUCCESS)
          {
              ctaGetText( ctahd, ret, (char *) errortext, 40);
              printf( "imgtReleaseBuffer failure: %s\n",errortext );
              exit( 1 );
           }
           break;
       } /* end of switch */
   }/* end of while */
}
```

imgtSendMessage

Sends a message to the management manager with attached data.

Prototype

DWORD **imgtSendMessage** (CTAHD ***ctahd***, IMGT_MESSAGE ****pmessage***, unsigned ***size,*** void ****pbuff***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
pmessage	Pointer to IMGT_MESSAGE structure.
size	Size of data block.

pbuff	Pointer to the primitive-specific data structure (as specified in the *IMGT_CONFIG structure* on page 38).
	`typedef struct IMGT_MESSAGE`
	`{`
	`  BYTE nai;  /* Network Access Identifier */`
	`  BYTE code;  /* Primitive code */`
	`  WORD nfas_group;  /* NFAS group number, only required for Configurations with duplicate NAI values on a board */`
	`  DWORD status;  /* Status or error for START/STOP */`
	`} IMGT_MESSAGE`

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	This return value means any of the following: • The ***pmessage*** argument is NULL. • The ***pbuff*** is NULL but ***size*** is non zero. • The size of the data exceeds MAX_IMGT_BUFFER_SIZE.
CTAERR_INVALID_HANDLE	The specified context handle is invalid.
IMGTERR_IMGT_NOT_STARTED	A management session has not been initialized.

Events

Event	Reason code
IMGTEVN_SEND_MESSAGE	The event value field contains one of the following reasons or an error code: SUCCESS IMGTERR_BUFFER_TOO_BIG The size of the buffer is too large.

Details

See *IMGT_CONFIG structure* on page 38 for a list of valid messages and their associated buffer structures.

Example

```
void MySendOOS( CTAHD ctahd, unsigned char nai, unsigned char Bchannel )
{
    IMGT_MESSAGE msg;
    char pdata[MAX_IMGT_BUFFER_SIZE];
    unsigned size;
    struct imgt_service *psvc;
    char *errortext="";

    /* send SERVICE_RQ on nai to set Bchannel out of service */
    msg.nai = nai;
    msg.code = IMGT_SERVICE_RQ;

    psvc = (struct imgt_service *) pdata;
    psvc->type = PREFERENCE_BCHANNEL;
    psvc->nai = msg.nai;
    psvc->Bchannel = Bchannel;
    psvc->status = OUT_OF_SERVICE;
    size = sizeof( struct imgt_service );

    ret = imgtSendMessage( ctahd, &msg, size, pdata );
    if (ret != SUCCESS)
    {
            ctaGetText( ctahd, ret, (char *) errortext, 40);
            printf( "imgtSendMessage failure: %s\n",errortext );
            exit( 1 );
    }
}
```

imgtStart

Starts the IMGT service on a board.

Prototype

DWORD **imgtStart** (CTAHD ***ctahd***, unsigned ***nai***, IMGT_CONFIG ****pconfig***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
nai	Network access identifier of the ISDN channel being monitored.

pconfig	Pointer to configuration structure:
	`typedef struct`
	`{`
	`  DWORD size;  /* Size of structure */`
	`  DWORD imgt_mask;  /* IMGT event mask */`
	`  DWORD trap_mask;  /* Trap event mask (not used) */`
	`  DWORD mon_mask;  /* Monitor event mask (not used) */`
	`  WORD nfas_group;  /* Only required for configurations with duplicate NAI values on a board */`
	`  BYTE pad[6];  /* Pads */`
	`} IMGT_CONFIG;`

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The specified context handle is invalid.
IMGTERR_INVALID_CONFIG	The IMGT_CONFIG data structure contains invalid fields (invalid fields are those that are not currently implemented).

Events

Event	Description
IMGTEVN_RCV_MESSAGE	An IMGT message with the code IMGT_STARTED_CO is returned in response to this command. The message status field contains one of the following return values or reasons: SUCCESS The management manager has been properly configured. The application only receives events for which it is registered. IMGTERR_BAD_NAI The NAI in the message structure is not valid. IMGTERR_NAI_IN_USE The IMGT manager has already been started on this NAI. IMGTERR_ISDN_NOT_STARTED The ISDN stack has not been started for this NAI.

Details

The NMS ISDN protocol stack must be started before the IMGT service is started. If the stack is stopped and started, IMGT messages remain as set, either enabled or disabled. See *IMGT_CONFIG structure* on page 38 for a list of valid event masks.

Example

```
void SetimgtConfiguration( CTAHD ctahd, unsigned char nai )
{
    IMGT_CONFIG config;
    DWORD ret;
    char *errortext="";

    memset(&config, 0, sizeof(IMGT_CONFIG));

    /* get service and restart events */
    config.imgt_mask = IMGT_SERVICE_MASK | IMGT_RESTART_MASK;

    /* start receiving events on nai */
    ret = imgtStart( ctahd, nai, &config );

    if (ret != SUCCESS)
    {
        ctaGetText( ctahd, ret, (char *) errortext, 40);
        printf( "imgtStart failure: %s\n",errortext );
        exit( 1 );
    }
    /* wait for IMGTEVN_RCV_MESSAGE */
    ...
}
```

See Also

imgtStop

imgtStop

Stops a management session initiated with **imgtStart**.

Prototype

DWORD **imgtStop** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext.**

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The specified context handle is invalid.

Events

Event	Description
IMGTEVN_RCV_MESSAGE	An IMGT message with the code IMGT_STOPPED is returned in response to **imgtStop**. The message status field contains SUCCESS. This message indicates that the management session has been stopped.

Example

```
void ResetimgtConfiguration( CTAHD ctahd )
{
   DWORD ret;
   char *errortext="";

   /* stop receiving any events on the appropriate context */
   ret =           imgtStop( ctahd );

   if (ret != SUCCESS)
   {
          ctaGetText( ctahd, ret, (char *) errortext, 40);
          printf( "imgtStop failure: %s\n",errortext );
          exit( 1 );
   }
   /* wait for IMGTEVN_RCV_MESSAGE */
   ...

}
```

See Also

imgtStart

6 Data structures

IMGT data structures

The IMGT service uses data structures to configure the ISDN protocol stack, send messages to the protocol stack, and receive messages from the protocol stack. This section presents the IMGT data structures in detail.

Structures and definitions directly related to functions are defined in the *imgtdef.h* header file. Structures and definitions related to primitive codes are defined in the *imgtsvc.h* header file.

Messaging structures

Messaging structures are used to convey messages and message data between the ISDN protocol stack and the application:

- IMGT_MESSAGE structure
- IMGT_MSG_PACKET structure
- IMGT_CONFIG structure

For more information on sending and receiving IMGT messages, refer to the *IMGT application overview* on page 17.

Primitive-related structures

The primitive-related structures section describes the primitives that can be sent in an IMGT message and the structures associated with these primitives. The tables in this section list those fields that are implemented for each variant.

The primitive-related structures include:

- IMGT_B_CHANNEL_STATUS_CO
- IMGT_B_CHANNEL_STATUS_RQ
- IMGT_D_CHANNEL_EST_CO
- IMGT_D_CHANNEL_EST_RQ
- IMGT_D_CHANNEL_REL_CO
- IMGT_D_CHANNEL_REL_RQ
- IMGT_D_CHANNEL_STATUS_CO
- IMGT_D_CHANNEL_STATUS_IN
- IMGT_D_CHANNEL_STATUS_RQ
- IMGT_REPORT_IN
- IMGT_RESTART_IN
- IMGT_SERVICE_CO
- IMGT_SERVICE_IN
- IMGT_SERVICE_RQ

IMGT_MESSAGE structure

A pointer to the IMGT_MESSAGE structure is passed to **imgtSendMessage** in the ***message*** argument. In this structure, the application specifies the NAI and the message to be sent. This information is expressed using one of the message primitives described in this section. The message primitive appears in the code field in this structure.

When the IMGT_MSG_PACKET structure is received by the application, the structure contains an IMGT_MESSAGE structure containing message data.

IMGT_MESSAGE is defined as follows:

```
typedef struct
{
  BYTE          nai;
  BYTE          code;
  WORD          nfas_group;
  DWORD         status;
} IMGT_MESSAGE;
```

IMGT_MSG_PACKET structure

Messages are sent from the protocol stack to the application in the IMGT_MSG_PACKET structure. The IMGT_MSG_PACKET structure contains an IMGT_MESSAGE substructure that contains the message and a data area containing the message header. A pointer to the IMGT_MSG_PACKET structure is included in the CTA_EVENT structure returned by **ctaWaitEvent**.

For more information about receiving Natural Access events, see the *Natural Access Developer's Reference Manual*.

IMGT_MSG_PACKET is defined as follows:

```
typedef struct IMGT_MSG_PACKET
{
    WORD data_size;                 /* Size of the data to follow       */
    WORD pad;
    IMGT_MESSAGE message;           /* IMGT message identification      */
    BYTE databuff[4];               /* Data included in packet >=0      */
} IMGT_MSG_PACKET;
```

IMGT_CONFIG structure

To initiate an IMGT session, an application must send an IMGT_CONFIG data structure to the IMGT manager. This data structure contains masks that indicate the message types the application expects to receive. IMGT_CONFIG is sent to the manager as a parameter in **imgtStart**. The data structure is defined as follows:

```
typedef struct IMGT_CONFIG
{
  DWORD size;          /* Size of this structure              */
  DWORD imgt_mask;     /* Mask for maintenance messages       */
  DWORD trap_mask;     /* Reserved for future use             */
  DWORD mon_mask;      /* Mask to monitor specific messages   */
  WORD  nfas_group;    /* NFAS group number                   */
  BYTE  bpads[6];      /* Only required for configurations with
                          duplicate NAI values on a board     */
} IMGT_CONFIG;
```

In this structure, imgt_mask, mon_mask, and trap_mask are bit masks that define which management events the application receives. The following table describes possible values for these masks:

Mask	Value	If set, the application receives...
imgt_mask	IMGT_SERVICE_MASK	A notification whenever a SERVICE message is received on the trunk.
imgt_mask	IMGT_RESTART_MASK	A notification whenever a RESTART message is received on the trunk.
imgt_mask	IMGT_D_CHANNEL_STATUS_MASK	A notification whenever the status of a D channel changes.
mon_mask	IMGT_REPORT_MASK	Some variant-specific messages.

IMGT_B_CHANNEL_STATUS_CO

Returned to the application in response to a B channel status request. The structure contains an attached buffer containing an imgt_service structure.

Related structure

```
struct imgt_service
  {
  BYTE type;              /* Either trunk or a individual B Channel  */
  BYTE nai;               /* Network Access Identifier               */
  BYTE BChannel;          /* B Channel                               */
  BYTE status;            /* Either IMGT_IN_SERVICE, IMGT_MAINTENANCE,
                             or IMGT_OUT_OF_SERVICE                  */
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	PREFERENCE_TRUNK PREFERENCE_BCHANNEL	x	x	x	x										x
nai	Network access identifier	x	x	x	x										x
Bchannel	B channel number	x	x	x	x										x
status	B channel status: IMGT_IN_SERVICE IMGT_MAINTENANCE IMGT_OUT_OF_SERVICE	x	x	x	x										x

Note: E10 is the same variant as 5ESS.

Details

For this message, the type field is always PREFERENCE_BCHANNEL.

IMGT_B_CHANNEL_STATUS_RQ

Indicates that the application has requested the status of a B channel.

Related structure

```
struct imgt_service
  {
  BYTE type;          /* Either trunk or a individual B Channel   */
  BYTE nai;           /* Network Access Identifier                */
  BYTE BChannel;      /* B Channel                                */
  BYTE status;        /* Either IMGT_IN_SERVICE, IMGT_MAINTENANCE,
                         or IMGT_OUT_OF_SERVICE                   */
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	Not applicable														
nai	Network access identifier	x	x	x	x										x
BChannel	B channel number	x	x	x	x										x
status	B channel status: Not applicable														

Note: E10 is the same variant as 5ESS.

Details

For this message, the type field must be PREFERENCE_BCHANNEL.

You must fill the nai and BChannel fields into the imgt_service data structure. All other fields are ignored.

IMGT_D_CHANNEL_EST_CO

Returned to the application in response to a request sent to the board to bring up the D channel, with an attached buffer containing an imgt_d_channel_control structure.

Related structure

```
struct imgt_d_channel_control
{
  DWORD result;              /* D channel control command result
                                (Success/Failure)       */
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
result	D channel control command result: IMGT_D_CHANNEL_CMD_SUCCESS IMGT_D_CHANNEL_CMD_BAD_LINK_STATE	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

Details

The result element of the returned structure indicates whether or not the command can be executed. IMGT_D_CHANNEL_CMD_SUCCESS indicates that the D channel was re-established. IMGT_D_CHANNEL_CMD_BAD_LINK_STATE indicates that the command cannot be executed.

IMGT_D_CHANNEL_EST_RQ

Used by the application to request that the D channel be brought up.

Related structure

```
struct imgt_d_channel_control
{
  DWORD result;              /* D channel control command result
                                (Success/Failure)       */
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
result	Not used.	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

IMGT_D_CHANNEL_REL_CO

Returned to the application in response to a request sent to the board to take down the D channel, with an attached buffer containing an imgt_d_channel_control structure.

Related structure

```
struct imgt_d_channel_control
{
  DWORD result;              /* D channel control command result
                                (Success/Failure)       */
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
result	D channel control command result: IMGT_D_CHANNEL_CMD_SUCCESS IMGT_D_CHANNEL_CMD_BAD_LINK_STATE	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

Details

The result element of the returned structure indicates whether or not the command can be executed. IMGT_D_CHANNEL_CMD_SUCCESS indicates that the D channel was re-established. IMGT_D_CHANNEL_CMD_BAD_LINK_STATE indicates that the command cannot be executed.

IMGT_D_CHANNEL_REL_RQ

Used by the application to request that the D channel be taken down.

Related structure

```
struct imgt_d_channel_control
{
  DWORD result;              /* D channel control command result
                                (Success/Failure)       */
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
result	Not used.	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

IMGT_D_CHANNEL_STATUS_CO

Returned in response to a D channel status request, with an attached buffer containing an imgt_d_channel_status structure.

Related structure

```
struct imgt_d_channel_status
{
  BYTE status;              /* Status (Released/Established/
                               Awaiting Establishment)       */
  BYTE bpad[3];
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
status	D channel status: IMGT_D_CHANNEL_STATUS_RELEASED IMGT_D_CHANNEL_STATUS_ESTABLISHED IMGT_D_CHANNEL_STATUS_AWAITING	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

IMGT_D_CHANNEL_STATUS_IN

Indicates the change of the D channel status. It is sent with an attached buffer containing an imgt_d_channel_status structure.

Related structure

```
struct imgt_d_channel_status
{
  BYTE status;              /* Status (Released/Established/
                               Awaiting Establishment)      */
  BYTE bpad[3];
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
status	D channel status: IMGT_D_CHANNEL_STATUS_RELEASED IMGT_D_CHANNEL_STATUS_ESTABLISHED IMGT_D_CHANNEL_STATUS_AWAITING	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

IMGT_D_CHANNEL_STATUS_RQ

Used by the application to request the status of a D channel.

Related structure

```
struct imgt_d_channel_status
{
  BYTE status;             /* Status(Released/Established/
                              Awaiting Establishment)    */
  BYTE bpad[3];
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
status	D channel status: Not applicable	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

Details

You can request D channel status for all variants.

IMGT_REPORT_IN

Indicates that the application has received a variant-specific message (or operation) that is stored in an attached buffer.

Related structure

```
struct imgt_report_hdr
{
  BYTE OperationID;                  /* Specifies the operation. */
  BYTE bpad[3];
};
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
OperationID	Defines these attached operations: IMGT_OP_ ID_SET_CALL_TAG IMGT_OP_ ID_TRFD_CALL_CLEARING			x											

Note: E10 is the same variant as 5ESS.

Details

The OperationID field defines which structure is included in the attached buffer. As the following structure description shows, each structure includes an imgt_report_hdr structure declaration. Only the NI2 variant is supported.

```
struct imgt_report_CallTag
{
  struct imgt_report_hdr hdr;
  BYTE Slot;
  BYTE pad[3];
  DWORD CallTag;
};
```

OperationID value	Description
IMGT_OP_ID_SET_CALL_TAG	The application received the result of **SetCallTag**, indicating that the Two B channel Transfer (TBCT) operation was successful. The result of the TBCT operation is stored in the attached imgt_report_CallTag structure. Slot: Identifies the line (timeslot) of the call that successfully invoked the TBCT operation. CallTag: Contains the value of the transferred call.
IMGT_OP_ID_TRFD_CALL_CLEARING	The application received the result of **TransferredCallClearing**, indicating that the call transferred by the TBCT operation was cleared. The result is stored in the attached imgt_report_CallTag structure. Slot: Not applicable. CallTag: Contains the value of the cleared and transferred call.

IMGT_RESTART_IN

Indicates that the ISDN stack has received a RESTART message from the remote end.

Related structure

```
struct imgt_restart
  {
  BYTE type;          /* Indicates restart either for the entire trunk
                         or a individual B Channel                  */
  BYTE nai;           /* NAI for RESTART indication                 */
  BYTE BChannel;      /* B Channel if type is PREFERENCE_BCHANNEL   */
  BYTE pad;
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	PREFERENCE_TRUNK PREFERENCE_BCHANNEL	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x		x x
nai	Network access identifier	x	x	x	x	x	x	x	x	x	x	x	x		x
BChannel	B channel number	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: E10 is the same variant as 5ESS.

Details

For the DMS variant, the receipt of a RESTART message for a single B channel resets the status of that B channel to in-service. A RESTART message for the entire interface resets the status of all the B channels on that trunk to in-service.

If type is set to PREFERENCE_TRUNK, ignore the BChannel field.

IMGT_SERVICE_CO

Returned in response to either a management or a call control request to put the channel in-service or out-of-service.

Related structure

```
struct imgt_service
  {
  BYTE type;                /* Either trunk or a individual B Channel   */
  BYTE nai;                 /* Network Access Identifier                */
  BYTE BChannel;            /* B Channel                                */
  BYTE status;              /* Either IMGT_IN_SERVICE, IMGT_MAINTENANCE,
                               or IMGT_OUT_OF_SERVICE                   */
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	PREFERENCE_TRUNK		x												
	PREFERENCE_BCHANNEL	x	x	x	x										x
nai	Network access identifier	x	x	x	x										x
BChannel	B channel number	x	x	x	x										x
status	B channel status: IMGT_IN_SERVICE IMGT_MAINTENANCE IMGT_OUT_OF_SERVICE	x	x	x	x										x

Note: E10 is the same variant as 5ESS.

Details

If the type field is set to PREFERENCE_TRUNK, ignore the BChannel field.

IMGT_SERVICE_IN

Indicates the ISDN stack has received a SERVICE message from the remote end.

Related structure

```
struct imgt_service
  {
  BYTE type;             /* Either trunk or a individual B Channel   */
  BYTE nai;              /* Network Access Identifier                */
  BYTE BChannel;         /* B Channel                                */
  BYTE status;           /* Either IMGT_IN_SERVICE, IMGT_MAINTENANCE,
                            or IMGT_OUT_OF_SERVICE                   */
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	PREFERENCE_TRUNK		x												
	PREFERENCE_BCHANNEL	x	x	x	x										x
nai	Network access identifier	x	x	x	x										x
BChannel	B channel number	x	x	x	x										x
status	B channel status: IMGT_IN_SERVICE IMGT_MAINTENANCE IMGT_OUT_OF_SERVICE	x	x	x	x										x

Note: E10 is the same variant as 5ESS.

Details

If the type field is set to PREFERENCE_TRUNK, ignore the BChannel field.

IMGT_SERVICE_RQ

Indicates that the application has requested a SERVICE message to be sent to the remote end.

Related structure

```
struct imgt_service
  {
  BYTE type;          /* Either trunk or a individual B Channel   */
  BYTE nai;           /* Network Access Identifier                */
  BYTE BChannel;      /* B Channel                                */
  BYTE status;        /* Either IMGT_IN_SERVICE, IMGT_MAINTENANCE,
                         or IMGT_OUT_OF_SERVICE                   */
  };
```

Field	Description	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
type	PREFERENCE_TRUNK		x												
	PREFERENCE_BCHANNEL	x	x	x	x										x
nai	Network access identifier	x	x	x	x										x
BChannel	B channel number	x	x	x	x										x
status	B channel status: IMGT_IN_SERVICE IMGT_MAINTENANCE IMGT_OUT_OF_SERVICE	x	x	x	x										x

Note: E10 is the same variant as 5ESS.

7 Demonstration program

IMGT service demonstration program: imgtdemo

Name

imgtdemo

Purpose

imgtdemo uses the IMGT service to set and monitor the status of D channels and B channels on an ISDN trunk. *imgtdemo* provides an example of a management application that uses ISDN Management service functions.

imgtdemo is multi-threaded and interactive. One thread is used for user input and the other sends commands and monitors events from the ISDN stack. Multiple instances of the program can be run to monitor multiple trunks. With a command-line option, you can specify the board, the NAI, the NFAS group number, or the NFAS configuration number coded in the application.

imgtdemo is supplied with the NMS ISDN Management software in executable format with its source code and *makefile*.

Featured functions

imgtReleaseBuffer, **imgtSendMessage**, **imgtStart**, **imgtStop**

Requirements

- One or more digital trunk interface boards
- Natural Access
- NMS ISDN software

Before you start the demonstration program, verify that:

- Natural Access is properly installed.
- The board is operating correctly.
- MVIP switching is correctly configured.
- The ISDN stack is properly running.

Usage

`imgtdemo` [***options***]

Where ***options*** represents one or more of the following:

Option	Meaning	Defaults
-a ***nai***	Network access identifier	0
-b ***board_number***	Board number	0
-g ***nfas_group***	NFAS group number	0
-n ***nfas_configNum***	Enable internal NFAS configuration	N/A
-h or -?	Help	N/A

Compilation

imgtdemo is supplied in executable format as well as source code. To recompile *imgtdemo*, do one of the following:

Under this OS...	Go to this directory...	Enter...
Windows	*\nms\ctaccess\demos\imgtdemo*	nmake
UNIX	*/opt/nms/ctaccess/demos/imgtdemo/*	make

For more information, refer to the *readme_isdn* file that was installed with Natural Access.

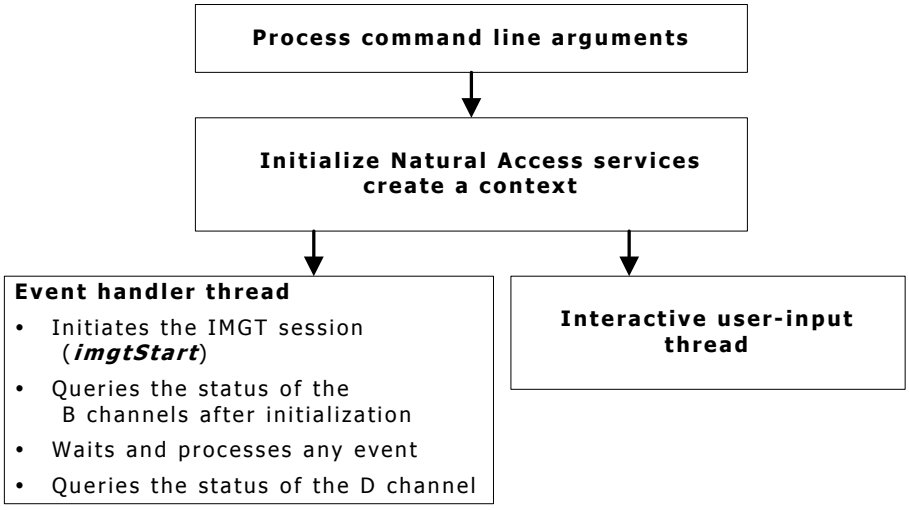
Files

imgtdemo consists of the following files:

File	Description
imgtdemo.c	The main application program with event processing functions.
imgtdemo.h	Header file for the IMGT demonstration program.

Details

The following illustration shows the architecture of the IMGT demonstration program:



imgtdemo program structure

The main program reads the command line options and opens both the ADI service and the IMGT service. *imgtdemo* then allocates a context containing the following information:

```
unsigned  NFAS_group          /* NFAS group number                      */
unsigned  board;              /* Board number                           */
unsigned  trunk;              /* Trunk number, from 0 - 3               */
unsigned  nai;                /* Network Access Identifier              */
unsigned  d_channel;          /* 0 - trunk not bearing D channel        */
                              /* 1 - trunk bearing D channel            */
unsigned  channel_status[30]; /* Store information about channel status */
```

If you use an NFAS configuration, one context is allocated for each trunk in the NFAS group. If you use the -n option at the command line, the program reads the internally hard-coded context structure. *imgtdemo* then spawns the user-input thread (**UserInput**) and the event-handler thread (**RunDemo**).

User-input thread

The *imgtdemo* user-input thread waits for the following user commands:

Command	Description
si	Starts the IMGT service.
sm	Sets the event mask interactively.
ss	Stops the IMGT service.
is ***low_channel high_channel*** [***nai***]	Puts a range of channels on ***nai*** in service. Specify ***nai*** for NFAS only.
tis [***nai***]	Puts the whole interface in service. Specify ***nai*** for NFAS only.
oos ***low_channel high_channel*** [***nai***]	Puts a range of channels on ***nai*** out of service. Specify ***nai*** for NFAS only.
toos [***nai***]	Puts the whole interface out of service. Specify ***nai*** for NFAS only.
sr ***low_channel high_channel*** [***nai***]	Requests the status for a range of channels on ***nai***. Specify ***nai*** for NFAS only.
sh	Shows the current status of B channels.
dsr	Requests the D channel status.
dsh	Shows the current D channel status.
help, h, ?	Displays this help screen.
quit, q	Exits the program.

If you use the oos, tis, toos, dsr, or sr commands, the program creates the messages based on passed arguments. The sh and dsh commands provide a snapshot of the status of all the B channels and the D channel (respectively) on the trunk or the whole NFAS group.

Example 1

To request a SERVICE message to put B channels from 1 through 23 out of service, enter the command:

```
oos 1 23
```

The message is sent to the IMGT manager.

Example 2

To show the status of the B channels on an E1 trunk, enter the sh command. The following information appears:

```
Show status on Board 1 Nai 0
Channels:    1 2 3 4 5 6 7 8 9 0  1 2 3 4 5 6 7 8 9 0  1 2 3 4 5 6 7 8 9 0
Status:      0 0 0 2 0 0 0 0 0 0  0 0 0 0 0 D 0 0 0 0  0 0 0 0 0 0 0 0 0 0
       0 = In Service, 1 = Maintenance,  2 = Out Of Service, D = D channel
```

Event-handler thread

The event-handler thread initiates the IMGT session by calling **imgtStart** with one of the following configuration masks:

This mask...	Enables...
0x00000007	All maintenance messages.
0x0001000	ISDN-specific messages. (For example, two B channel transfer notification.)

The event-handler thread also performs the following tasks:

- Queries the status of all the B channels in the trunk or NFAS group.
- Queries the status of the D channel.
- Waits and processes any event.

8 Errors and events

NMS ISDN Management service errors

The IMGT service error codes are presented in two tables:

- Alphabetically, by error code name, including a description of the problem and a possible solution.
- Numerically, by hexadecimal value, decimal value, and error name.

All Natural Access functions return a status code. If the return code is not SUCCESS (0), it is an error code indicating that the function has failed and the reason for the failure.

IMGT service error codes are defined in the include file *imgtdef.h*. The error codes are prefixed with IMGT. Error codes can also appear in the value field of a DONE event. Use the CTA_IS_ERROR macro to determine if a value is an error.

Because the IMGT service is a Natural Access service, IMGT functions can also receive Natural Access errors and reasons. Refer to the *Natural Access Developer's Reference Manual* for listings and descriptions of the Natural Access errors and reasons.

Alphabetical error summary

The following table alphabetically lists the IMGT service errors. All errors are 32 bit.

Error name	Hexadecimal	Decimal	Description
IMGTERR_BAD_NAI	0x00181001	1576961	Invalid NAI specification.
IMGTERR_BUFFER_TOO_BIG	0x00181004	1576964	Buffer is too big.
IMGTERR_DRIVER_ERROR	0x00181007	1576967	Driver error.
IMGTERR_IMGT_NOT_STARTED	0x00181016	1576982	Management session has not been initialized.
IMGTERR_INVALID_BUFFER	0x00181008	1576968	Invalid buffer.
IMGTERR_INVALID_CONFIG	0x00181002	1576962	Invalid IMGT_CONFIG field.
IMGTERR_ISDN_NOT_STARTED	0x00181015	1576981	ISDN stack is not started.
IMGTERR_NAI_IN_USE	0x00181006	1576966	NAI is already configured.
IMGTERR_NO_UP_BUFFER	0x00181003	1576963	No incoming buffer.
IMGTERR_RCV_BUFFER_TOO_BIG	0x00181005	1576965	Incoming buffer is too big.

Numerical error summary

The following table numerically lists the IMGT service errors:

Hexadecimal	Decimal	Error Name
0x00181001	1576961	IMGTERR_BAD_NAI

Hexadecimal	Decimal	Error Name
0x00181002	1576962	IMGTERR_INVALID_CONFIG
0x00181003	1576963	IMGTERR_NO_UP_BUFFER
0x00181004	1576964	IMGTERR_BUFFER_TOO_BIG
0x00181005	1576965	IMGTERR_RCV_BUFFER_TOO_BIG
0x00181006	1576966	IMGTERR_NAI_IN_USE
0x00181007	1576967	IMGTERR_DRIVER_ERROR
0x00181008	1576968	IMGTERR_INVALID_BUFFER
0x00181015	1576981	IMGTERR_ISDN_NOT_STARTED
0x00181016	1576982	IMGTERR_IMGT_NOT_STARTED

NMS ISDN Management service events

All events in the Natural Access environment are represented by a C data structure, as shown in the generic CTA_EVENT below:

```
typedef struct
    {
    DWORD id;           /* Event id (LIBEVN_xxx in 'libdef.h')          */
    CTAHD ctahd;        /* Context handle                               */
    DWORD timestamp;    /* Timestamp                                    */
    DWORD userid;       /* Use-supplied id                              */
    DWORD size;         /* Size of buffer if buffer is not NULL         */
                        /* Otherwise, may contain event-specific data   */
    void *buffer;       /* Buffer pointer                               */
    DWORD value;        /* Event status or event-specific data          */
    DWORD objHd;        /* Service client-side object handle            */
    } CTA_EVENT;
```

This structure, returned by **ctaWaitEvent**, informs the application which event occurred on which context and provides additional information specific to the event. The LIB prefix in LIBEVN indicates the association between events and specific NMS services. For example, the IMGT prefix is associated with the IMGT service.

Because the IMGT service is a Natural Access service, IMGT functions can also receive Natural Access events. Refer to the *Natural Access Developer's Reference Manual* for listings and descriptions of the Natural Access events.

The event structure contains the following fields:

Field	Description
id	An event code defined in the library header file. All IMGT events are prefixed with IMGTEVN_ (for example, IMGTEVN_SOMETHING_HAPPENED).
ctahd	The context handle (returned from **ctaCreateContext**).
timestamp	The time when the event was created, in milliseconds since midnight, January 1, 1970, modulo 49 days. The resolution for board events is ten milliseconds.
userid	The user-supplied ID. This field is unaltered by Natural Access and facilitates asynchronous programming. Its purpose is to correlate a context with an application object and context when events occur.
size	The size (in bytes) of the area pointed to by the buffer. If the buffer is NULL, this field may be used to hold an event-specific value.
buffer	A pointer to data returned with the event. The field contains an application process address. The event size field contains the actual size of the buffer.
value	A reason code or an error code. This field contains an event-specific value.
objHd	The call handle, if the event concerns a particular call. If the event concerns the line and not a particular call, objHd is NULL.

IMGT service event summary

The following events are specific to the NMS ISDN Management service:

Event	Hexadecimal	Decimal	Description
ISDNEVN_SEND_MESSAGE	0x00182003	1581059	Message has been sent and failed.
ISDNEVN_RCV_MESSAGE	0x00182005	1581061	Message was received from the trunk.

Index