Doug and

Dialogic®

Making Innovation Thrive™

Dialogic® NaturalAccess™ GR303 and V5 Libraries Developer's Manual

October 2009 64-0517-01

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
9000-62083-10	October 2001	MVH, Platform support for DLCP 1.0 Beta
9000-62083-11	December 2001	MVH, Platform support for DLCP 1.0
9000-62083-12	January 2002	MVH, Platform support for DLCP 1.1
9000-62083-13	November 2002	SRG, Natural Access 2003-1 Beta
9000-62083-14	April 2003	MVH, Natural Access 2003-1
9000-62083-15	April 2004	MCM, Natural Access 2004-1
64-0517-01	October 2009	LBG, NaturalAccess R9.0
Last modified: September 14, 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™ GR303 and V5 Libraries Developer's Manual* explains how to use the NaturalAccess GR303 library and the GR303 V5 library to develop applications that use DLCP protocol stacks running on NaturalAccess interface platforms.

This document is intended for developers of telephony and voice applications who are using NaturalAccess. This document defines telephony terms where applicable, but assumes that you are familiar with telephony concepts 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 DLCP overview

Digital loop carrier protocols

Digital loop carrier protocols (DLCPs) outline digital mechanisms for bringing a wide range of services to users through twisted-pair copper phone lines. Remote digital terminals use DLCP protocols to communicate with central office (CO) switches. By utilizing DLCP protocols, new network access gateways can easily interface with CO switches, allowing service providers to implement new services with minimal impact to the installed network infrastructure.

The GR303 and V5.2 DLCP protocols enable carriers to use location-specific remote concentrators in the local loop while increasing the reach of individual switches in the network. In these scenarios, gateways must inter-operate with various types of network equipment while enabling carriers to provide new services. The use of proven protocol stacks enables carriers to deploy innovative services without performing extensive testing on each switch in the network.

Access303 and Exchange303 GR303 support

GR303 defines a generic set of requirements for integrated access systems. These requirements specify an open interface that provides mix and match flexibility between the local digital switches (LDSs) and the remote digital terminals (RDTs), and between RDTs and element management systems.

GR303 requirements were defined by Telcordia to eliminate the proprietary interfaces that were common among access systems. GR303 promotes increased network architecture flexibility by providing a consistent approach to deploying access technologies in the network. GR303 interfaces reduce the cost of deployment by supporting a flexible service concentration.

In the GR303 scheme, DS1 facilities connecting to the digital switch are assigned and managed through a timeslot management channel (TMC), while remote operations functions are supported over an embedded operations channel (EOC). A data link processor (DLP) common control module terminates the GR303 EOC and TMC interfaces from the switch.

Aztek's GR303 protocol stack (Access303 and Exchange303) software enables gateways and remote digital terminals in North America to interact with the central office. Access303 and Exchange303 provide a complete TMC and EOC (timeslot management channels and embedded operations channels) implementation of the GR303 specification for both the IDT and RDT side of the interface.

Access AV5 V5.2 protocol support

Access network (AN) applications can use a V5.2 protocol stack to connect to local exchange (LE) equipment. In addition, equipment can use the V5.2 protocol for the following access methods:

- Analog telephone access
- Other digital or analog access for semi-permanent connections without associated out-of-band signalling information

The V5.2 protocol can use up to sixteen 2048 kbps links. For analog access on the LE side, the application converts signaling from PSTN ports into a functional part of the V5.2 protocol for signalling to the AN side. For ISDN applications, a control protocol within the V5.2 protocol defines a method for exchanging individual functions and messages required for coordinating call control procedures on the local exchange.

The V5.2 protocol includes the following features for supporting increased traffic and dynamic link allocation:

- A bearer channel connection protocol that the local exchange can use to establish and tear down bearer connections (identified by the signaling information) on demand.
- A link control protocol for multi-link management that allows applications to control link identification, link blocking, and link failure conditions.
- A protection protocol (operated on two separate data links for security reasons) for managing communication channel switching in the event of link failures.

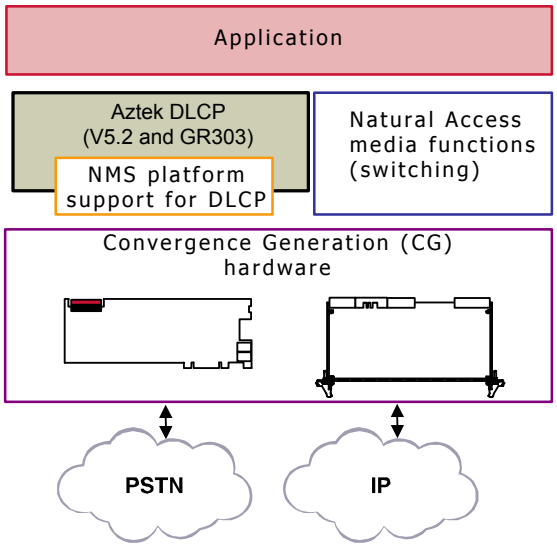
Aztek AV5 software is designed to meet the requirements of access network equipment and supports the V5.1, V5.2, 1st Edition, and 2nd Edition ETSI standards.

NMS DLCP software

The NMS DLCP software provides tools for developing applications that use DLCP protocol stacks running on NMS telecommunications interface platforms. For example, network equipment providers can develop access gateway solutions that connect voice over cable networks to existing PSTNs to make new services accessible through legacy networks. Using the GR303 and V5.2 protocols allows carriers to quickly integrate such gateways into their networks.

The NMS DLCP software also provides tools that equipment providers can use to create solutions for entering new markets. For example developers can use NMS DLCP software tools to develop access gateways that transport voice over fixed wireless networks and provide new services to subscribers in remote areas.

The following illustration is an overview of the NMS DLCP development environment:



Frequently used acronyms

Acronym	Description
AN	Access network. The portion of a public switched network that connects access nodes to individual subscribers. It is the last link in a network between the customer premises and the first point of connection to the network infrastructure, usually a point of presence (PoP) or central office (CO).
BCC	Bearer channel connection protocol. A protocol that allows the local exchange (LE) to instruct the access network (AN) to allocate bearer channels on demand, either singly or in multiple instances.
DLCP	Digital loop carrier protocols. Protocols that make use of digital techniques to bring a wide variety of services to users through twisted-pair copper phone lines.
DS1	Digital Signal 1. A serial digital signal transmission format in which 24 duplex voice circuits are time division multiplexed into one 1.544 Mbps T1 digital circuit. Also referred to as T1.
EOC	Embedded operations channel. A channel provided on telecommunications facilities to support remote management operations such as provisioning, maintenance, protection switching, and alarm surveillance.
ETSI	European Telecommunications Standards Institute.
IDT	Integrated digital terminal. A virtual network element that represents the operation of a protocol (for example GR303) within a switch.
ISDN	Integrated services digital network. A standard for providing voice and data telephone service with all-digital transmission and message-based signaling.
HDLC	High-level data link control. A link layer protocol for point-to-point and multiple-port communications.
LAPD	Link access procedures for the D-channel. A protocol for communication at the data link layer.
LAPV5	Link access procedures for the V5.2 protocol. A sub-layer of the V5.2 link layer that provides reliable, end-to-end communication paths between the AN and LE for exchanging PSTN signaling information and other V5.2 control information.
LDS	Local digital switch.
LE	Local exchange. Exchange in which subscriber lines terminate, having access to other exchanges, to national trunk networks, and to the central office (CO). A local exchange carrier (LEC) is a common carrier that operates local exchanges in a given geographical area.
PSTN	Public switched telephone network. The domestic telecommunications network commonly accessed by ordinary telephones, PBX trunks, and data communications facilities.
RDT	Remote digital terminal. The digital loop carrier multiplexer at the end of the digital loop carrier that is closest to the network interface. According to the GR303 protocol, an RDT provides the side of the interface devoted to subscribers. When communicating with the PSTN, a media gateway performs functions of the RDT.
TMC	Timeslot management channel. A channel used for timeslot allocation and de-allocation and for assigning and managing facilities connected to an LDS.

4 NMS DLCP development environment

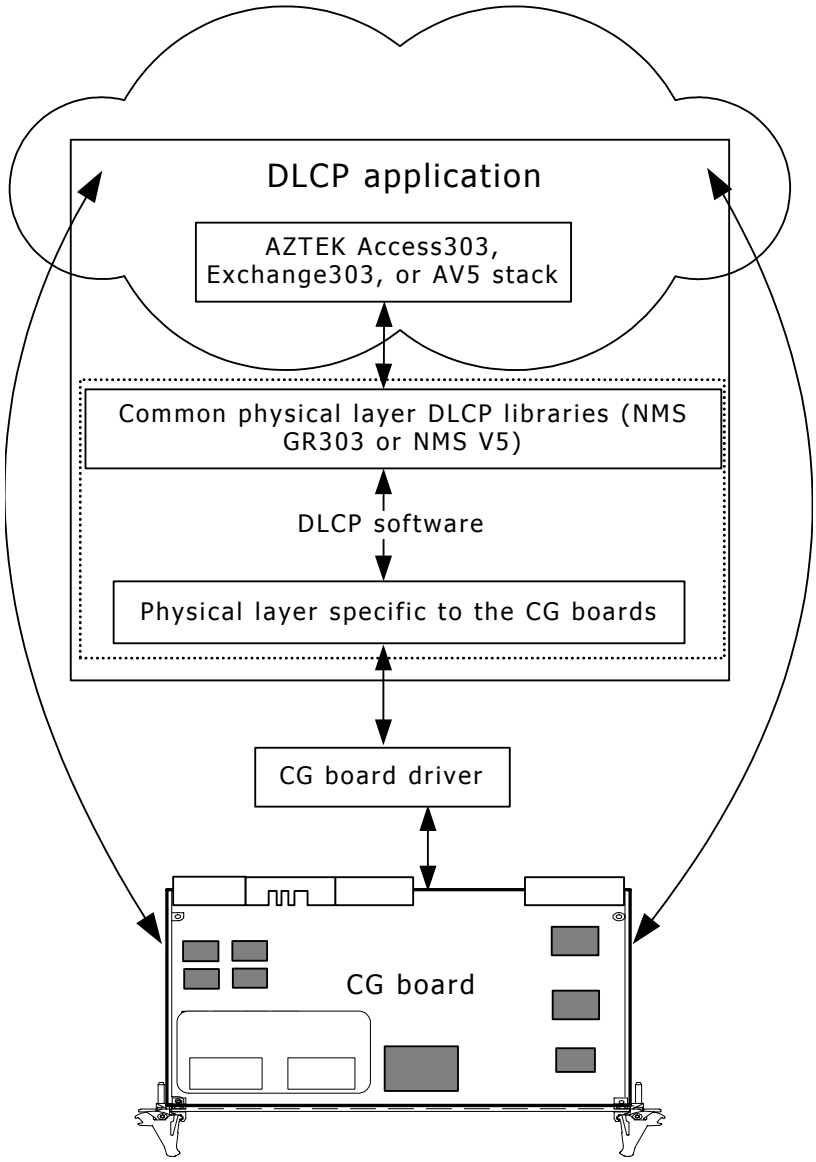
NMS DLCP system overview

To develop DLCP applications, you must have the following hardware and software:

- One or more CG 6000, CG 6000C, CG 6100C, or CG 6500C boards
- Natural Access
- NMS DLCP software (installed when you install Natural Access)
- Aztek Access303, Exchange303 (GR303) or AV5 (V5.2) protocol stack software
- C/C++ development environment

DLCP (GR303 or V5) systems are based on several components. CG boards provide a physical layer hardware platform. The DLCP software provides GR303 and V5.2 protocol-specific interfaces so that NMS hardware can support protocol Layer 1 functionality. Applications also use Aztek's Access303, Exchange303, or AV5 protocol stack software to integrate GR303 or V5.2 compliant upper layer interfaces.

The following illustration provides an overview of the NMS DLCP system environment:



To perform CG board switching, media processing, or signaling processing tasks, applications can invoke functions from Natural Access. For example, GR303 and V5 protocols require dynamic switching between timeslots on a per-call basis. Applications can do this by using functions from the Switching (SWI) service. In addition, the GR303 protocol defines a 16-state robbed-bit signaling option for transmitting call-processing messages between the RDT and the IDT. Applications must use the ADI service to control CAS robbed-bit signaling required by the GR303 protocol.

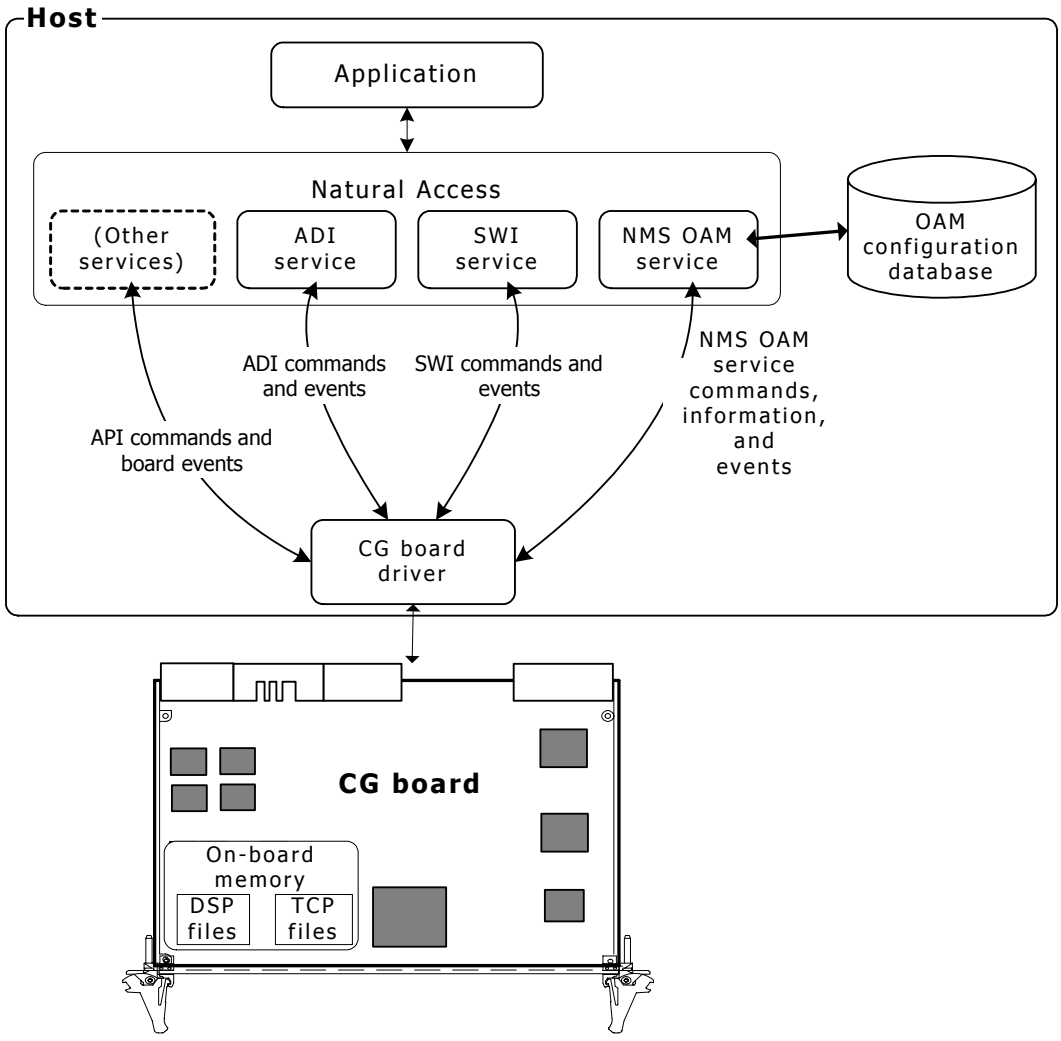
The application may also need other NMS software, such as ISDN protocol software to terminate ISDN calls, or NMS Fusion software to implement VoIP gateway functionality. The system can also use additional NMS boards to perform as ISDN PRI or BRI line cards.

Natural Access components

Natural Access is a complete software development environment for voice applications. It provides a standard set of functions grouped into logical services. Each service has a standard programming interface. The Natural Access components include:

- Natural Access development environment that provides services for performing call control, system configuration, media processing, and other functions.
- NMS OAM software that configures, administers, and maintains the system.
- OAM configuration files that describe how the board is set up and initialized.
- Runtime software and drivers for controlling CG boards.
- One or more trunk control programs (TCPs) that allow applications to communicate with the telephone network using the signaling schemes (or protocols) used on the trunk.
- NMS DLCP software

The following illustration shows the Natural Access software environment:



DLCP applications require the following services:

Service	Description
SWI	Switching service that provides a set of functions for controlling CT bus switch blocks on MVIP and H.100/H.110 compliant switching devices. Switching service functions control the switch block to make or break connections, send patterns, sample data, query, reset, configure, and run diagnostics on different parts of the MVIP switching device. Refer to the *Switching Service Developer's Reference Manual* for more information.
ADI	Media service for NMS CG boards. NMS GR303 applications require this service to perform CAS ABCD robbed-bit signaling. Refer to the *ADI Service Developer's Reference Manual* for more information.

If your application requires functionality provided by any other Natural Access services, you must also install and initialize these services.

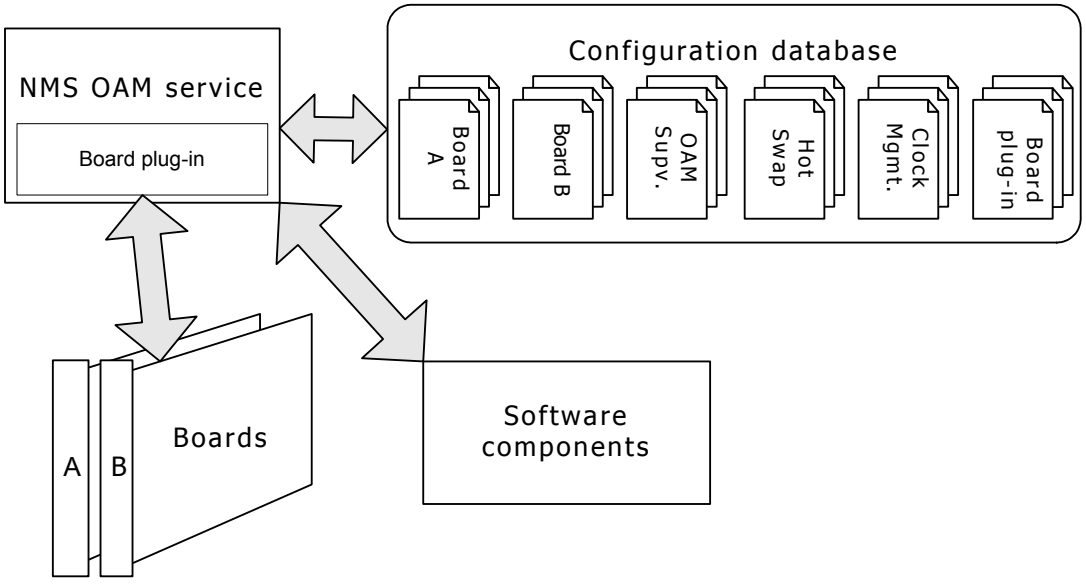
NMS OAM

NMS OAM manages and maintains system resources. These resources include hardware components (including CG boards) and low-level board management software modules (such as the Hot Swap process).

You can perform the following tasks using NMS OAM:

- Create, edit, delete, and query the configuration of a component.
- Start (boot), stop (shut down), and test a component.
- Receive notifications from components.

NMS OAM maintains a database containing records of configuration information for each component as shown in the following illustration:



Information in this configuration database consists of parameters and values, expressed as keyword name/value pairs (for example, Encoding = muLaw). You can query the NMS OAM database for keyword values in any component. Keywords and values can be added, modified or deleted.

Use the *oamsys* utility to initialize your boards based on the information in the configuration files. *oamsys* transfers all software modules specified in the system configuration file (and the board keyword files referenced by it) to the boards on the system. *oamsys* also performs other configuration tasks as needed. In addition, NMS OAM provides the following utilities:

- *oammon* allows you to monitor boards for errors and other events.
- *oamcfg* allows you to change system information or board parameters while the system is running.

For more information about NMS OAM and NMS OAM utilities, refer to the *NMS OAM System User's Manua*l.

NMS OAM configuration files

An NMS OAM system configuration file and the board keyword files it references provide information that tell the *oamsys* configuration program how to configure the boards on the system. When you set up your system, edit these files to specify configuration information for all boards in your system.

Several sample board keyword files are included with the DLCP software. These sample board keyword files show keyword settings for systems that use different types of CG boards. For more information, refer to *Customizing OAM configuration files* on page 27.

Runtime software

CG runtime software consists of runfiles (also known as downloadable modules or DLMs), a core file stored in non-volatile memory (Flash memory), and DSP files. The runfile is the basic low-level software that a CG board requires to operate. DSP files enable the CG board on-board DSPs to perform certain tasks, such as DTMF signaling, voice recording, voice playback, CAS robbed-bit signaling, and HDLC processing.

Several runfiles and DSP files are installed with Natural Access. You specify which files to use for your configuration in the board keyword file. When NMS OAM boots a board, the runfile and DSP files are transferred from the host into on-board memory.

Trunk control programs

To provide call control capabilities on PSTN ports, applications associate a telephony protocol with each port. On CG boards, telephony protocols are supported as trunk control programs (TCPs) that are loaded when the board is initialized. When applications start a protocol, TCPs enable the application to use the call control functions associated with that protocol. NMS provides TCPs for most standard telephone line interfaces, and also provides a NOCC (no call control) TCP for applications that do not perform call control or that manage line interfaces manually. For more information about Natural Access TCP software, refer to the *NMS CAS for Natural Call Control Developer's Manual*.

NMS DLCP software

The NMS DLCP software is installed when you install Natural Access. NMS DLCP software consists of the following components:

Component	Description
NMS GR303 library	Enables remote digital terminal (RDT) or integrated digital terminal (IDT) applications to perform GR303 protocol stack operations through a simplified API.
NMS V5 library	Enables access network (AN) applications to perform V5.2 protocol stack operations through a simplified API.
nmstool303 and *nmstoolv5* demonstration programs	Enable developers to test various functions from the NMS GR303 and NMS V5 libraries and to test the system integration with the Aztek Access303, Exchange303, or AV5 protocols stacks.
Sample configuration files	Sample NMS OAM configuration files for configuring CG boards for use with the NMS GR303 and NMS V5 libraries.

Aztek protocol stacks

Aztek protocol software implements the upper layers of the V5.2 and GR303 protocols. Aztek provides the following products that can be obtained separately:

- Access303 (RDT side of the GR303 protocol) and Exchange303 (IDT side of the GR303 protocol) software
- AV5 (V5.2 protocol) software

Access303 and Exchange303 protocol software

Aztek Access303 and Exchange303 software support the remote digital terminal (RDT) and integrated digital terminal (IDT) sides of a GR303 interface, as defined in GR303-CORE and GR303-IMD. This software includes the time slot management channel (TMC) and link access procedures for the D-channel (LAPD) protocols.

Access303 and Exchange303 also provide embedded operations channel (EOC) functions of a convergence layer, a remote operations service element (ROSE) layer, a common management interface service layer (CMISE), and path protection functionality. Access303 and Exchange303 also include the AIM-303 (*aim303*), an integration tool for the protocol software.

Access303 and Exchange303 are based on a layered software architecture. Layer 2 supports the LAPD protocol. When communicating between an RDT and an IDT, LAPD sends messages to, or receives messages from, the physical layer (Layer 1) provided by the application. When communicating with the rest of the RDT or IDT application, LAPD exchanges messages with the upper layer.

Access303 meets GR303-CORE and GR303-IMD specifications and the GR303 interface specifications for switch vendors. Because Access303 and Exchange303 are independent from the underlying network architecture, applications that integrate with the Access303 and Exchange303 protocol stacks can be based on digital loop carrier (DLC), hybrid fiber coax (HFC), wireless local loop (WLL), voice over the internet protocol (VOIP), or any other access network architecture.

The Access303 and Exchange303 products are provided as source code that can be ported to a variety of hardware and operating system platforms for which an ANSI C compiler is available. The Access303 and Exchange303 stacks can be built in either library or executable form.

AV5 protocol software

Aztek AV5 software implements the access network (AN) end of an ETSI V5 protocol. AV5 provides five Layer 3 protocols, one Layer 2 protocol, and layer management. Supported Layer 3 protocols include:

Protocol	Description
PSTN	Supports signaling for analog ports.
BCC	Supports dynamic timeslot assignment.
Control	Supports synchronized application of provisioning data, port blocking, and restart.
Link control	Provides link blocking and coordination of link numbering.
Protection switching	Supports standby protection of the signaling channels.

AV5 supports the Layer 2 protocol LAPV5, which is a LAPD-like protocol specifically adapted for V5 interfaces. AV5 also provides frame relay capabilities for ISDN D-channels at Layer 2. The layer management entity supports a set of management capabilities including data link management, E1 link management, protection management, and the provisioned database capabilities. These management features allow higher level management of the Layer 3 protocol entities, and thus provide for simplified system management.

AV5 is a complete implementation of the V5.1 and V5.2 protocol with additional features and tools that has been proven compliant with industry standards through testing with the ETSI abstract test suite (ATS).

AV5 interacts with an AN system in the following areas:

- An RTOS abstraction interface for which Aztek provides sample code.
- A PSTN nationalization sample task.
- APIs for connections through a TSI.
- APIs with Layer 1 and Layer 1 management functions.
- APIs with system management.

The AV5 product is provided as source code that can be ported to a variety of hardware and operating system platforms for which an ANSI C compiler is available. The AV5 stack can be built as either a library or an executable.

For detailed information about Aztek software components, refer to the Aztek documentation.

Developing IDT, RDT, and AN applications

To use the NMS DLCP software to develop IDT (integrated digital terminal), RDT (remote digital terminal), or AN (access network) applications, perform the following steps:

Step	Description	For more information, refer to...
1	Install Natural Access. (The NMS DLCP software is included in the Natural Access installation.)	Natural Access installation booklet.
2	Install Aztek's Access303, Exchange303, or AV5 protocol stack software.	Aztek Access303, Exchange303, or AV5 documentation.
3	Build the Access303, Exchange303, or AV5 protocol stack as a library.	Access303, Exchange303, or AV5 integration guides.
4	Build the Aztek *aim303* or *aimv5* utility as an executable.	Access303, Exchange303, or AV5 integration guides, Aztek Integration Menu AIM user guides.
5	Build the *nms303tool* or *nmsv5tool* demonstration program as an executable.	*Building program executables* on page 156.
6	Create a mapping for board, trunk, and channel locations between DLCP libraries (NMS GR303 or NMS V5) and Aztek protocol stack libraries (Access303, Exchange303, or AV5).	*Building program executables* on page 156.
7	Configure the CG hardware (through NMS OAM system configuration and board keyword files) and boot the boards.	*Configuring the system overview* on page 27.
8	Connect CG boards to the peer (RDT), IDT, or LE side of the interface.	CG board installation and developer's manual.
9	Start the *nms303tool* or *nmsv5tool* demonstration program.	*Using the demonstration programs* on page 155.
10	Start the *aim303* or *aimv5* utility.	Aztek Integration Menu AIM user guides
11	Use options from the NMS DLCP demonstration programs (*nms303tool* or *nmsv5tool*) to execute functions at the physical layer of the GR303 or V5.2 protocol stack, and use options from the Aztek AIM utilities (*aim303* or *aimv5*) to execute functions at the upper layer of the GR303 or V5.2 protocol stack.	Using the demonstration programs and the Aztek Integration Menu AIM User Guides.

You can create RDT, IDT, or AN applications by adding functionality to the *nms303tool* or *nmsv5tool* demonstration programs, or by creating your own applications.

Building Access303, Exchange303, or AV5 libraries

Several architectural approaches are available for designing the RDT, IDT, or AN applications. This manual describes scenarios where the Access303, Exchange303, or AV5 software are built as libraries and then linked to applications. You do not need to make changes to the Access303, Exchange303, or AV5 source files to perform basic system integration.

Refer to the Access303, Exchange303, or AV5 integration guide for more information about building Aztek protocol stack libraries.

Using the AIM utilities

Aztek Integration Menu (AIM) utilities, *aim303* and *aimv5*, are installed with Access303, Exchange303, and AV5 software. These utilities allow you to test and integrate Access303, Exchange303, and AV5 functions into the RDT or AN applications.

Each AIM utility is an interactive console application that performs a variety of functions, such as calling API functions, configuring task and message tracing, and displaying status information. Each AIM utility operates as a stand-alone process that communicates with a dedicated task on the Aztek protocol stack to pass commands to the stack.

The AIM utilities are provided as source code. You must compile the applications before using them. For more information about the *aim303* and *aimv5* utilities, refer to the *Aztek Integration Menu AIM-303 User Guide* or *Aztek Integration Menu AIM-V5 User Guide*.

Using nms303tool and nmsv5tool

The *nms303tool* and *nmsv5tool* demonstration programs provide a way of verifying that the NMS DLCP software is installed and operating correctly. Use these utilities to:

- Test individual NMS GR303 or NMS V5 functions or combinations of the functions.
- Expose working examples of GR303 functions.
- Provide a working software example of an integrated application that uses the Aztek and DLCP software libraries.

nms303tool and *nmsv5tool* are menu-driven interactive programs. Enter commands to exercise different integration options while simultaneously running the Access303, Exchange303, or AV5 protocol stacks and the *aim303* or *aimv5* utilities. *nms303tool* and *nmsv5tool* are provided as source code. You must compile the applications before using them.

5 Configuring the system

Configuring the system overview

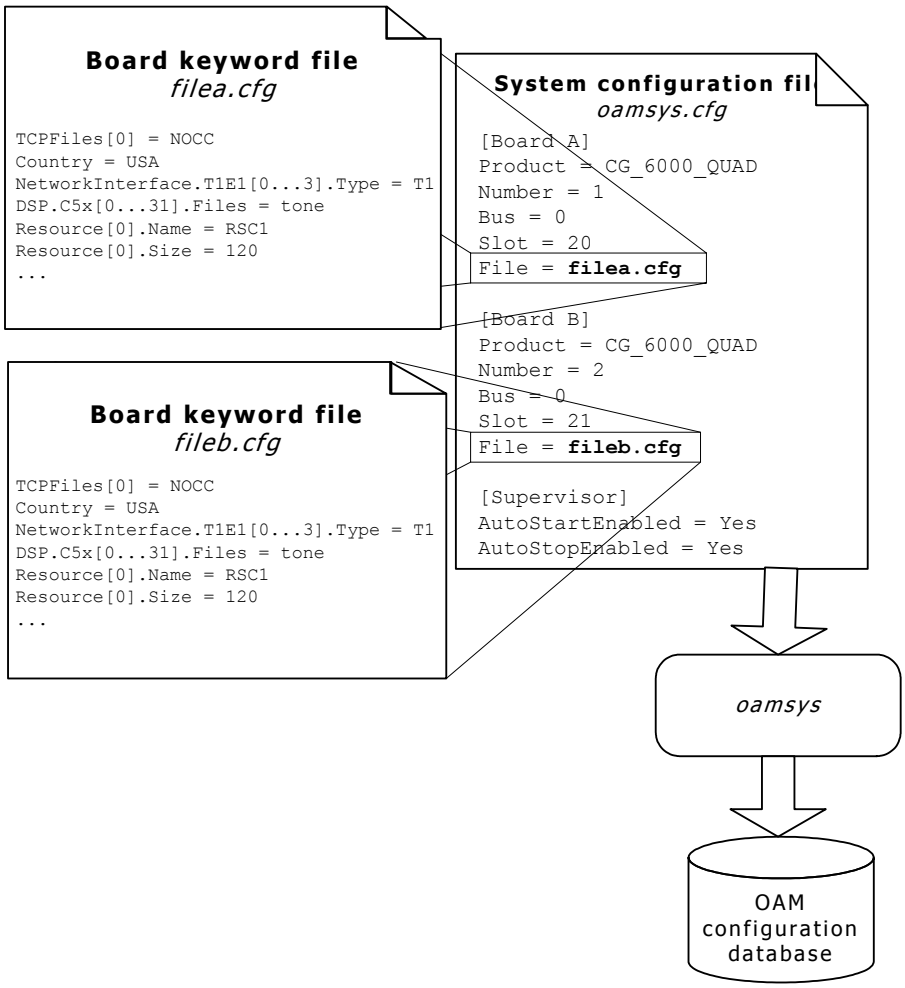
The NMS hardware that provides the physical interface for DLCP is designed for flexibility to support a wide variety of applications. You must configure the hardware to suit DLCP applications. You can customize the hardware configuration by creating configuration files (system configuration and board keyword files) or editing the sample configuration files that accompany DLCP software.

Step		Description
1	Create or edit an NMS OAM system configuration file and board keyword files.	Specify system and board keyword settings that optimize the CG hardware configuration for the DLCP environment.
2	Run the *oamsys* utility with the edited NMS OAM configuration files.	Updates the NMS OAM configuration database and configure the CG hardware on the system. For more information, refer to the *NMS OAM System User's Manual*.
3	Run one of the demonstration programs *nms303tool* or *nmsv5tool.*	Verifies that the DLCP software can initialize a Layer 1 for GR303 or V5.2 protocols, set up provisioning, start interfaces, perform transmit and receive HDLC operations, and report trunk alarms and performance statistics.

Customizing OAM configuration files

To configure and start the boards, specify configuration parameters in the board keyword file. Then reference the board keyword file for each board in another file, the system configuration file. When you run *oamsys*, it creates a record for each board in the OAM database and stores the parameters and values of the board. *oamsys* then starts the board, configured as described in the database.

The following illustration shows the relationship between the board keyword files and the system configuration file:



Modifying sample CG board keyword files

The DLCP software provides several sample board keyword files that demonstrate how to use NMS OAM board keywords for particular types of system configurations. Edit a sample board keyword file to specify appropriate configuration information for your system, including the line interface type. For detailed information about customizing CG board keyword files, refer to the board's hardware documentation. The following table lists the DLCP sample board keyword files:

Board keyword file	Board type	Description
cg6gr303.cfg	CG 6000 or CG 6000C	Configuration file for T1 interfaces used on GR303 systems.
cg6v5.cfg	CG 6000 or CG 6000C	Configuration file for E1 interfaces used in V5.2 systems.
cg61gr303.cfg	CG 6100C	Configuration file for T1 interfaces used on GR303 systems.
cg61v5.cfg	CG 6100C	Configuration file for E1 interfaces used in V5.2 systems.

Board keyword file	Board type	Description
cg65gr303.cfg	CG 6500C	Configuration file for T1 interfaces used on GR303 systems.
cg65v5.cfg	CG 6500C	Configuration file for E1 interfaces used in V5.2 systems.

CG board settings specified in the board configuration files include:

- Line interface type (T1/E1)
- Board clocking configuration
- On-board resource management
- Downloadable software modules

Configuring T1 or E1 line interfaces

To configure the T1 (for GR303) or E1 (for V5) interfaces, ensure that the following keywords display in the board keyword file:

OAM keyword	GR303 (T1) values	V5.2 (E1) values	Specifies...
NetworkInterface.T1E1[***x***].Type	T1	E1	Trunk type for each trunk on the board.
NetworkInterface.T1E1[***x***].Impedance	DSX1	G703_120_OHM | G703_75_OHM	Type of cable connecting a CG board to the telephone network.
NetworkInterface.T1E1[***x***].LineCode	AMI | B8ZS | AMI_ZCS | AMI_BELL |AMI_DDS| AMI_GTE	HDB3 | AMI	Ones density maintenance method used on the trunk line to maintain a clear channel transmission.
NetworkInterface.T1E1[***x***].FrameType	ESF	CEPT	T1 or E1 trunk framing format for the current board(s) or current trunk(s).
NetworkInterface.T1E1[***x***].SignalingType	CAS	RAW	How voice and signaling information is routed to and from the E1 or T1 trunk and DSP resources.

For all keywords in the table, ***x*** is a 0-based trunk number or range of trunk numbers as defined by NMS OAM. For CG 6000 or CG 6000C boards, ***x*** can be an integer between zero and three. For CG 6100C or CG 6500C boards, ***x*** can be an integer between zero and 15.

Note: When setting the NetworkInterface.T1E1[***x***].Type keyword, you must specify all trunks as either T1 trunks or E1 trunks. Do not specify more than one trunk type.

For more information, refer to the T1 and E1 trunk channels section in this manual.

Creating a system configuration file

When your board keyword file(s) are complete, create a system configuration file that describes the overall configuration of your system and references the board keyword file for each board. The system configuration file is typically named *oamsys.cfg*. Refer to the *NMS OAM System User's Manual* for information on the syntax and structure of *oamsys.cfg*.

Note: You can use the *oamgen* utility (included with the NMS OAM software) to create a sample system configuration file for your system. The system configuration file created by *oamgen* may not be appropriate for your configuration. You may need to make further modifications to the file before running *oamsys* to configure your boards based on the file. For more information about *oamgen*, refer to the *NMS OAM System User's Manual.*

When necessary, you can modify the CG board bus and slot number to include multiple boards in a system. Each board entry in the system configuration file must contain the following keywords:

Keyword	Description
Product	Board product type
Number	Logical board number
Bus	PCI Bus number of the CG board
Slot	PCI Slot number of the CG board

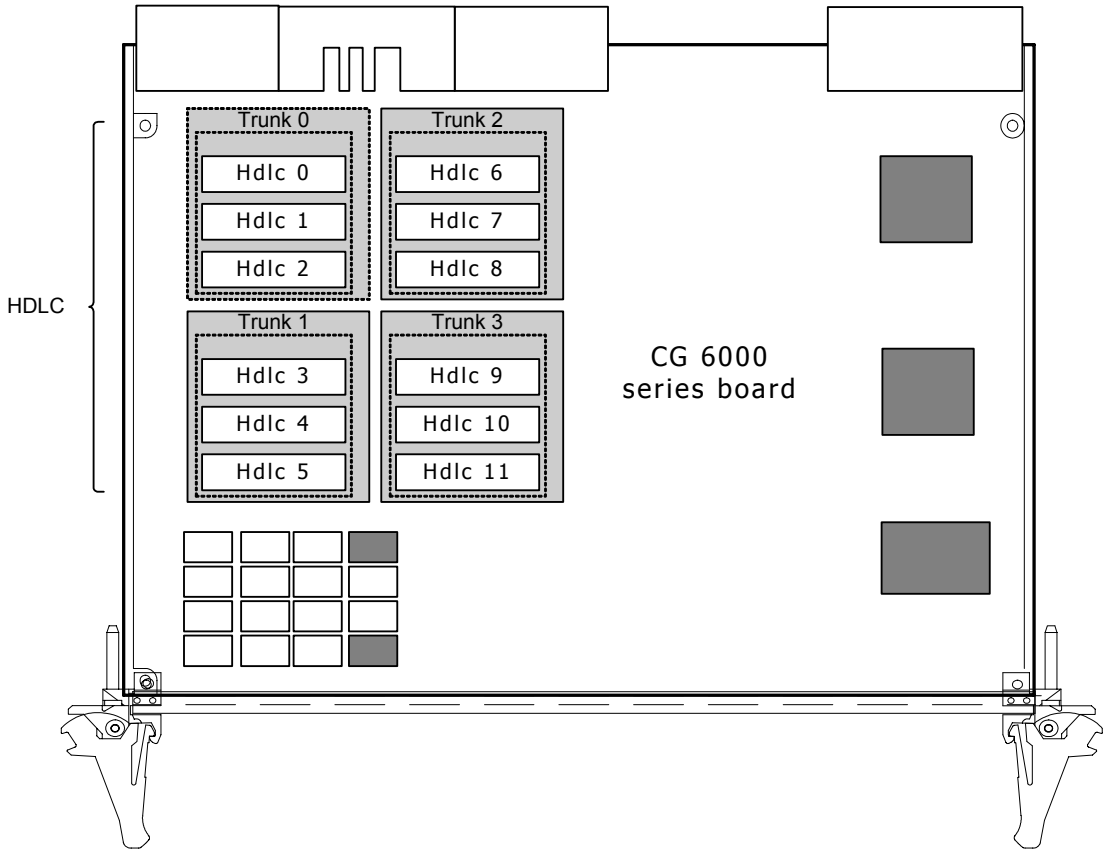
To obtain CG board bus and slot numbers, run the *pciscan* utility. For more information about *pciscan*, refer to the *NMS OAM Service System Users's Manual*.

After you customize the individual board keyword files and create the system configuration file, run the *oamsys* utility to configure the boards as specified in the configuration files. When you run the *oamsys* utility, it creates NMS OAM database records based on the contents of the specified system configuration file and board keyword files. It then directs NMS OAM to start the boards and configure them according to the specified parameters.

Configuring HDLC processing on CG 6000/C boards

CG 6000 and CG 6000C boards use the on-board HDLC controllers to perform HDLC data processing. Each CG board T1 or E1 trunk is associated with a single framer instance. On CG 6000 and CG 6000C boards each framer instance is hard-wired to three HDLC instances.

Applications can use functions from the NMS GR303 or NMS V5 libraries to dynamically configure and operate these HDLC instances once the board environment is booted. The HDLC instances on CG 6000 and CG 6000C boards are organized as shown in the following illustration:

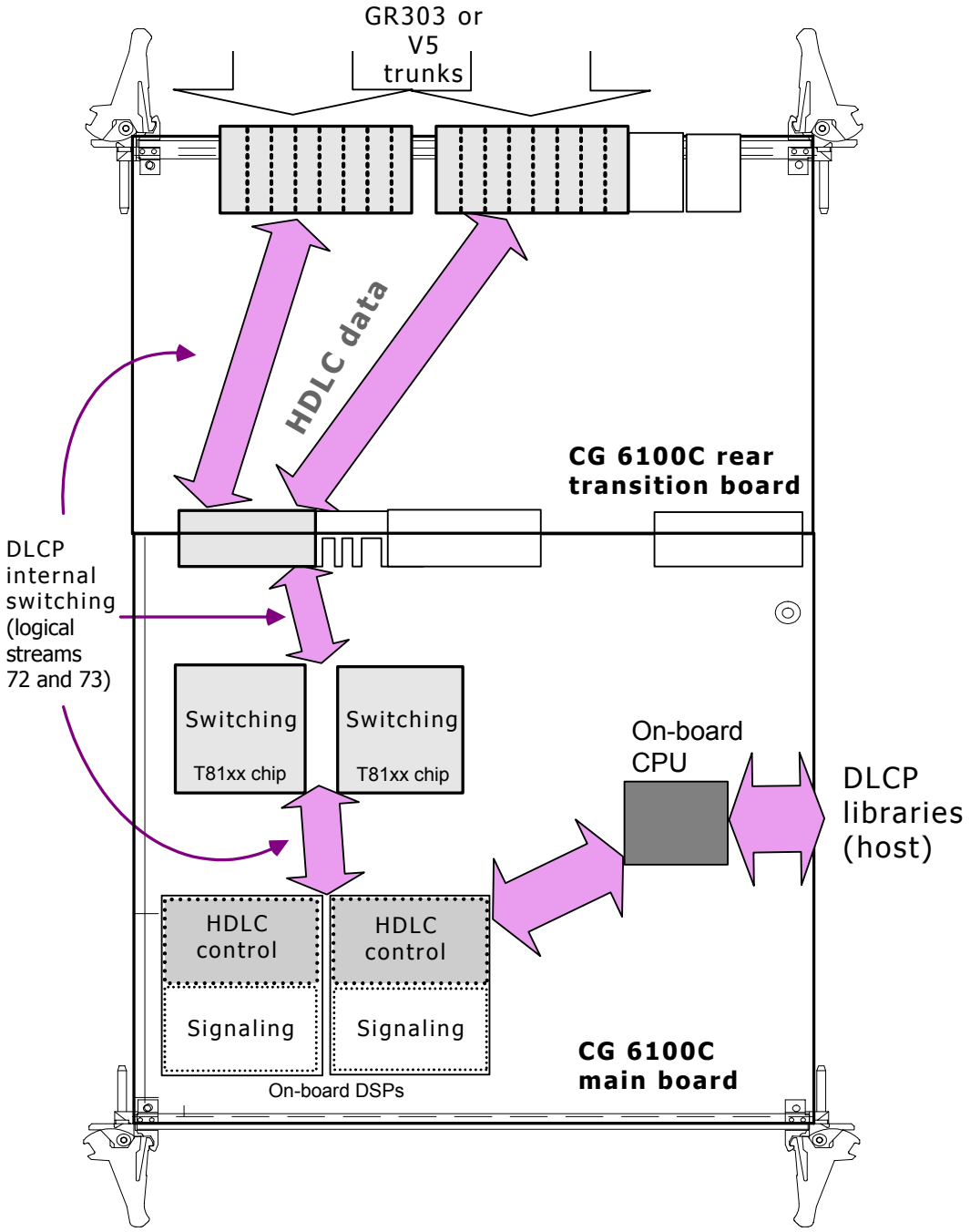


Refer to *CG 6000 and CG 6000C board keyword files* on page 34 for examples of board keywords used to configure these boards for NMS GR303 and NMS V5.

Configuring HDLC processing on CG 6100C and CG 6500C boards

On CG 6100C and CG 6500C boards, HDLC processing is performed through the on-board DSP cores. The application must configure in advance any DSP resources necessary for HDLC processing. The NMS GR303 and V5 libraries automatically perform any necessary switching between associated T1/E1 interfaces and DSP cores to set up HDLC channels. When an application creates an HDLC channel for a particular T1 or E1 link, the NMS GR303 library or NMS V5 library switches the HDLC data from a line interface to an assigned DSP core for processing. When the application destroys the HDLC channel, the associated switch connections are restored to their initial state.

The following illustration shows switching performed internally by the NMS GR303 library to support up to 16 T1 trunks on a CG 6100C board with four DSP cores. In the example, the trunks can support up to 32 HDLC channels of data. Because each DSP core can process up to 16 channels of data, two on-board DSP cores are required to process the HDLC data.



Caution:	Once the application has provisioned at least one HDLC channel, it should not reset the CG 6100C switch block until all channels are destroyed. If the application resets the board switch block while any HDLC channels are active, any associated HDLC channel connections are destroyed.

Use the *showcx95* utility to verify internal switch connections set up by the NMS GR303 or NMS V5 libraries.

Setting up DSP resources for HDLC on CG 6100C and CG 6500C boards

When configuring the CG 6100C or CG 6500C board, configure DSP resources to support the maximum number of HDLC channels that your application requires. Consider the following information when you set up the board:

- Each DSP core can handle up to 16 HDLC channels.
- DSP cores configured for HDLC data processing cannot be used for any other kind of data processing. That is, any DSP core used for processing HDLC channels must be dedicated to that task.
- Before configuring the CG 6100C boards, determine the number of DSP cores you need to provide HDLC processing resources and which on-board DSPs you will use. For example, if your application requires up to 48 HDLC channels, you must configure at least three DSP cores (48/16 = 3) to provide HDLC resources.

To configure the DSP HDLC processing resources, use the board keywords described in the sections that follow.

Note: The keywords described in the following sections are mandatory for any CG 6100C or CG 6500C board that performs HDLC processing.

DSP board keywords

Include the following board keywords in every CG 6100C or CG 6500C board keyword file that will provide DSP resources for HDLC data processing:

Keyword	Description
DSP.HDLC.Enable	Enables the stream for HDLC running on the DSPs. Set this keyword to YES to run DLCP on the CG 6100C board. When set to YES, the CG 6100C switch model supports two additional streams, 72 and 73, for full-duplex DSP-based HDLC control.
DSP.C5x[***x***].CmdQStart = 0xEA00 DSP.C5x[***x***].CmdQSize = 0x00fe DSP.C5x[***x***].DataInQStart = 0xEB00 DSP.C5x[***x***].DataInQSize = 0xA00 DSP.C5x[***x***].DspOutQStart = 0xF500 DSP.C5x[***x***].DspOutQSize = 0xA80	Changes the memory map of the specified DSP(s). You can modify the DSP range to indicate the appropriate DSP core, but you cannot change the HDLC values provided in the sample board keyword files. **Note:** Without these keywords, a DSP core cannot process more than four channels of HDLC.
DSP.C5x[***x***].XLaw = ***compand***	Enables or disables companding on the specified DSP cores. Disable companding for any DSP cores dedicated to the HDLC processing. Specify the range of DSPs you want to dedicate to HDLC processing. For example, to configure DSP cores three, four, and five for HDLC companding, specify the following (this (disables companding): `DSP.C5x[3..5].Xlaw = NO_LAW`

Keyword	Description
DSP.C5x[***x***].Files = ***DPMfile***	Specifies digital signal processor function modules (DPMs) loaded to CG board DSPs. For example: `DSP.c5x[x].Files = hdlc`
DSP.c5x[***x***].Libs = ***libfile***	Specifies which TCP(s) to load to the CG onboard processors. For example: `DSP.c5x[x].Libs = cg6klibu hdlc_lib`
DLMFiles[0] = ***dlmfile***	Specifies an optional runtime component (modular extension to the core file) to be transferred to the board by the configuration file. For example: `DLMFiles[0] = cg6krun`

For all applicable keywords, ***x*** is a zero-based range of DSP numbers. That is, 0 is associated with the first physical on-board DSP.

Caution:	The DSP keyword values shown are specifically required for configuring the CG 6100C and CG 6500C DSP resources needed for HDLC processing. Use only the allowed values shown for these keywords.

For more information about setting up DSP resources for other processing tasks (for example, signaling, echo cancellation, or IVR functions), refer to the board's installation and developer's manual or to the *NMS OAM System User's Manual*.

Other board-specific keywords

On CG 6100C and CG 6500C boards, you must also specify an appropriate call trunk control program (TCP) to run on the board. If the application does not want to perform on-board call control, it should set the TCPFiles keyword to nocc. This setting specifies that the board does not reserve DSP resources for performing call control.

Refer to and *CG 6100C board keyword files* on page 37 and *CG 6500C board keyword files* on page 40 for examples of board keywords used to configure these boards for NMS GR303 and NMS V5.

CG 6000 and CG 6000C board keyword files

This topic provides examples of:

- Keywords used to configure CG 6000 and CG 6000C T1 boards for NMS GR303
- Keywords used to configure CG 6000 and CG 6000C E1 boards for NMS V5

GR303 configuration

The following board keyword file configures a CG 6000 or CG 6000C T1 board to run the GR303 protocol:

```
Clocking.HBus.ClockSource                          = NETWORK
Clocking.HBus.ClockSourceNetwork                   = 1
TCPFiles                                           = nocc

NetworkInterface.T1E1[0..3].Type                   = T1
NetworkInterface.T1E1[0..3].Impedance              = DSX1
NetworkInterface.T1E1[0..3].LineCode               = B8ZS
NetworkInterface.T1E1[0..3].FrameType              = ESF
NetworkInterface.T1E1[0..3].SignalingType          = CAS

DSPStream.VoiceIdleCode[0..3]                      = 0x7F
DSPStream.SignalIdleCode[0..3]                     = 0x00

DSP.C5x[0..31].Libs[0]                             = cg6kliba
DSP.C5x[0..31].XLaw                                = A_LAW
DSP.C5x[1..31].Files                               = voice tone dtmf echo rvoice callp \
                                                ptf wave oki ima gsm_ms g726 mf
DSP.C5x[0].Files                                   = qtsignal tone dtmf echo NULL NULL NULL

Resource[0].Name                                   = RSC1
Resource[0].Size                                   = 120
Resource[0].TCPs                                   = nocc
################################################################
Resource[0].Definitions = ( dtmf.det_all & echo.ln20_apt25 & ptf.det_2f & tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )

DLMFiles[0]                                        = cg6krun

DebugMask                                          = 0
```

V5.2 configuration

The following board keyword file configures a CG 6000 or CG 6000C T1 board to run the V5.2 protocol:

```
#*******************************************************************************************
#  NOTE: Adjust clocking mode accordingly
#*******************************************************************************************
Clocking.HBus.ClockSource                       = NETWORK
Clocking.HBus.ClockSourceNetwork                = 1
#*******************************************************************************************
#  NOTE: Adjust idle code and TCP Files in Resource Definition string for the country you
#  are using.
#*******************************************************************************************
TCPFiles                                        = nocc

DSPStream.VoiceIdleCode[0..3]                   = 0xD5
DSPStream.SignalIdleCode[0..3]                  = 0x0D

NetworkInterface.T1E1[0..3].Type                = E1
NetworkInterface.T1E1[0..3].Impedance           = G703_120_OHM
NetworkInterface.T1E1[0..3].LineCode            = HDB3
NetworkInterface.T1E1[0..3].FrameType           = CEPT
NetworkInterface.T1E1[0..3].SignalingType       = RAW

DSP.C5x[0..31].Libs[0]                          = cg6kliba
DSP.C5x[0..31].XLaw                             = A_LAW
DSP.C5x[1..31].Files                            = voice tone dtmf echo rvoice callp \
                                                ptf wave oki ima gsm_ms g726 mf
DSP.C5x[0].Files                                = qtsignal tone dtmf echo NULL NULL NULL

Resource[0].Name                                = RSC1
Resource[0].Size                                = 120
Resource[0].TCPs                                = nocc
################################################################
Resource[0].Definitions                         = ( dtmf.det_all & echo.ln20_apt25 & \
ptf.det_2f & tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )

DLMFiles[0]                                     = cg6krun

DebugMask                                       = 0
```

For more information about customizing the DLCP software, refer to *Integration overview* on page 43. For more information about NMS OAM, refer to the *NMS OAM System User's Manual*. Refer to the board's installation and developer's manual for more information about CG board keyword files.

CG 6100C board keyword files

This topic provides examples of:

- Keywords used to configure CG 6100C T1 boards for NMS GR303
- Keywords used to configure CG 6100C E1 boards for NMS V5

GR303 configuration

The following board keyword file configures a CG 6100C T1 board to run the GR303 protocol:

```
#*******************************************************************************************
#  NOTE: Adjust clocking mode accordingly
#*******************************************************************************************
Clocking.HBus.ClockSource                       = NETWORK
Clocking.HBus.ClockSourceNetwork                = 1
#*******************************************************************************************
#  NOTE: Adjust idle code and TCP Files in Resource Definition string for the country you
#  are using.
#*******************************************************************************************
NetworkInterface.T1E1[0..15].Type                = T1
NetworkInterface.T1E1[0..15].Impedance           = DSX1
NetworkInterface.T1E1[0..15].LineCode            = B8ZS
NetworkInterface.T1E1[0..15].FrameType           = ESF
NetworkInterface.T1E1[0..15].SignalingType       = CAS

DSPStream.VoiceIdleCode[0..15]                   = 0x7F
DSPStream.SignalIdleCode[0..15]                  = 0x00
DSP.HDLC.Enable                                 = YES

DSP.C5x[0,1,4..9].Libs                          = cg6klibu
DSP.C5x[2,3].Libs                               = cg6klibu hdlc_lib
DSP.C5x[2,3].XLaw                               = NO_LAW
DSP.C5x[0,1,4..9].XLaw                          = MU_LAW
DSP.C5x[0,1].Files                              = 8tsignal
DSP.C5x[2,3].Files                              = hdlc
DSP.C5x[2,3].CmdQStart                          = 0xEA00
DSP.C5x[2,3].CmdQSize                           = 0x00fe
DSP.C5x[2,3].DataInQStart                       = 0xEB00
DSP.C5x[2,3].DataInQSize                        = 0xA00
DSP.C5x[2,3].DspOutQStart                       = 0xF500
DSP.C5x[2,3].DspOutQSize                        = 0xA80

Resource[0].Name                                = RSC1
Resource[0].Size                                = 24
Resource[0].TCPs                                = nocc
#########################################################################
Resource[0].Definitions = ( dtmf.det_all & echo.ln20_apt25 & ptf.det_2f & tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )

Resource[0].DSPs = 4 5 6 7 8 9
#########################################################################
```

```
#  The following resource definitions are REQUIRED in DLCP configurations
#############################################################################
Resource[1].Name                                = hdlc
Resource[1].Size                                = 32
Resource[1].TCPs                                = nocc
Resource[1].Definitions                         = ( hdlc.duplex )
Resource[1].DSPs                                = 2 3

DLMFiles[0]                                     = cg6krun
DebugMask                                       = 0
```

V5.2 configuration

The following board keyword file configures a CG 6100C E1 board to run the V5.2 protocol:

```
#*****************************************************************************************
#  NOTE: Adjust clocking mode accordingly
*****************************************************************************************
Clocking.HBus.ClockSource                       = NETWORK
Clocking.HBus.ClockSourceNetwork                = 1
#*****************************************************************************************
#  NOTE: Adjust idle code and TCP Files in Resource Definition string for the country you
#  are using.
#*****************************************************************************************
NetworkInterface.T1E1[0..15].Type                = T1
NetworkInterface.T1E1[0..15].Impedance           = DSX1
NetworkInterface.T1E1[0..15].LineCode            = B8ZS
NetworkInterface.T1E1[0..15].FrameType           = ESF
NetworkInterface.T1E1[0..15].SignalingType       = CAS

DSPStream.VoiceIdleCode[0..15]                   = 0x7F
DSPStream.SignalIdleCode[0..15]                  = 0x00
DSP.HDLC.Enable                                 = YES

DSP.C5x[0,1,4..9].Libs                          = cg6klibu
DSP.C5x[2,3].Libs                               = cg6klibu hdlc_lib
DSP.C5x[2,3].XLaw                               = NO_LAW
DSP.C5x[0,1,4..9].XLaw                          = MU_LAW
DSP.C5x[0,1].Files                              = 8tsignal
DSP.C5x[2,3].Files                              = hdlc
DSP.C5x[2,3].CmdQStart                          = 0xEA00
DSP.C5x[2,3].CmdQSize                           = 0x00fe
DSP.C5x[2,3].DataInQStart                       = 0xEB00
DSP.C5x[2,3].DataInQSize                        = 0xA00
DSP.C5x[2,3].DspOutQStart                       = 0xF500
DSP.C5x[2,3].DspOutQSize                        = 0xA80

Resource[0].Name                                = RSC1
Resource[0].Size                                = 24
Resource[0].TCPs                                = nocc
#####################################################################
Resource[0].Definitions         = ( dtmf.det_all & echo.ln20_apt25 & ptf.det_2f & \
tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )
```

```
Resource[0].DSPs = 4 5 6 7 8 9
########################################################################
#  The following resource definitions are REQUIRED in DLCP configurations
########################################################################
Resource[1].Name                                = hdlc
Resource[1].Size                                = 32
Resource[1].TCPs                                = nocc
Resource[1].Definitions                         = ( hdlc.duplex )
Resource[1].DSPs                                = 2 3

DLMFiles[0]                                     = cg6krun
DebugMask                                       = 0
```

For more information about customizing the DLCP software, refer to *Integration overview* on page 43. For more information about NMS OAM, refer to the *NMS OAM System User's Manual*. Refer to the board's installation and developer's manual for more information about CG board keyword files.

CG 6500C board keyword files

This topic provides examples of:

- Keywords used to configure CG 6500C T1 boards for NMS GR303
- Keywords used to configure CG 6500C E1 boards for NMS V5.

GR303 configuration

The following board keyword file configures a CG 6500C T1 board to run the GR303 protocol:

```
Clocking.HBus.ClockMode                          = STANDALONE
Clocking.HBus.ClockSource                        = OSC
Clocking.HBus.ClockSourceNetwork                 = 1
#*******************************************************************************
#  T1 configuration
#*******************************************************************************
DSPStream.VoiceIdleCode[0..15]                    = 0x7F
DSPStream.SignalIdleCode[0..15]                   = 0x00

NetworkInterface.T1E1[0..15].Type                 = T1
NetworkInterface.T1E1[0..15].Impedance            = DSX1
NetworkInterface.T1E1[0..15].LineCode             = B8ZS
NetworkInterface.T1E1[0..15].FrameType            = ESF
NetworkInterface.T1E1[0..15].SignalingType        = CAS

#   Required in CG6500 DLCP configurations
DSP.HDLC.Enable                                   = YES

#   DSP settings required by HDLC
DSP.C5x[2,3].Libs                                 = cg6klibu hdlc_lib
DSP.C5x[2,3].XLaw                                 = NO_LAW
DSP.C5x[2,3].Files                                = hdlc
DSP.C5x[2,3].CmdQStart                            = 0xEA00
DSP.C5x[2,3].CmdQSize                             = 0x00fe
DSP.C5x[2,3].DataInQStart                         = 0xEB00
DSP.C5x[2,3].DataInQSize                          = 0xA00
DSP.C5x[2,3].DspOutQStart                         = 0xF500
DSP.C5x[2,3].DspOutQSize                          = 0xA80
#  Place everything else on the remaining DSPs
DSP.C5x[0,1,4..95].Libs                           = cg6klibu
DSP.C5x[0,1,4..95].XLaw                           = MU_LAW
#*******************************************************************************
DSP.C5x[0..1].Files                               = 8tsignal
#*******************************************************************************
#  Resource management
#*******************************************************************************
Resource[0].Name                                  = RSC1
Resource[0].Size                                  = 120
Resource[0].TCPs                                  = nocc
#*******************************************************************************
# Before modifying this resource definition string refer to the CG6500C
# Installation and Developers Manual.
#*******************************************************************************
Resource[0].Definitions  = ( dtmf.det_all & echo.ln20_apt25 & ptf.det_2f & tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
```

```
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )
Resource[0].DSPs = \
4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33\
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 \
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 \
92 93 94 95
#########################################################################
#  The following resource definitions are REQUIRED in DLCP configurations
#########################################################################
Resource[1].Name                                = hdlc
Resource[1].Size                                = 32
Resource[1].TCPs                                = nocc
Resource[1].Definitions                         = ( hdlc.duplex )
Resource[1].DSPs                                = 2 3
DLMFiles[0]                                     = cg6500run
DebugMask                                       = 0x0
```

V5.2 configuration

The following board keyword file configures a CG 6500C E1 board to run the V5.2 protocol:

```
#*************************************************************************************************
#  NOTE: Adjust clocking mode accordingly
#*************************************************************************************************
Clocking.HBus.ClockSource                        = NETWORK
Clocking.HBus.ClockSourceNetwork                 = 1
TCPFiles                                         = nocc
#*************************************************************************************************
#  NOTE: Adjust idle code and TCP Files in Resource Definition string for the country you
#  are using.
#*************************************************************************************************
DSPStream.VoiceIdleCode[0..15]                   = 0xD5
DSPStream.SignalIdleCode[0..15]                  = 0x0D
NetworkInterface.T1E1[0..15].Type                = E1
NetworkInterface.T1E1[0..15].Impedance           = G703_120_OHM
NetworkInterface.T1E1[0..15].LineCode            = HDB3
NetworkInterface.T1E1[0..15].FrameType           = CEPT
NetworkInterface.T1E1[0..15].SignalingType       = RAW
#   Required in CG6500 DLCP configurations
DSP.HDLC.Enable                                  = YES
#   Configure the HDLC DSPs
DSP.C5x[2..4].Libs                               = cg6kliba hdlc_lib
DSP.C5x[2..4].XLaw                               = NO_LAW
DSP.C5x[2..4].Files                              = hdlc
DSP.C5x[2..4].CmdQStart                          = 0xEA00
DSP.C5x[2..4].CmdQSize                           = 0x00fe
DSP.C5x[2..4].DataInQStart                       = 0xEB00
DSP.C5x[2..4].DataInQSize                        = 0xA00
DSP.C5x[2..4].DspOutQStart                       = 0xF500
DSP.C5x[2..4].DspOutQSize                        = 0xA80
#   Configure remaining call control/voice DSPs
DSP.C5x[0,1,5..95].Libs                          = cg6kliba
DSP.C5x[0,1,5..95].XLaw                          = A_LAW
DSP.C5x[0,1].Files                               = 8tsignal
Resource[0].Name                                 = RSC0
Resource[0].Size                                 = 120
Resource[0].TCPs                                 = nocc
#########################################################################
# Before modifying this resource definition string refer to the CG6500
# Installation and Developers Manual.
#########################################################################
Resource[0].Definitions         = ( dtmf.det_all & echo.ln20_apt25 & ptf.det_2f &\
tone.gen & \
callp.gnc & ptf.det_4f & \
( (rvoice.rec_mulaw & rvoice.play_mulaw) | \
(rvoice.rec_alaw & rvoice.play_alaw) | \
```

```
(rvoice.rec_lin & rvoice.play_lin) | \
(voice.rec_16 & (voice.play_16_100 | voice.play_16_150 | voice.play_16_200)) | \
(voice.rec_24 & (voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec_32 & (voice.play_32_100 | voice.play_32_150 | voice.play_32_200)) | \
(voice.rec_64 & (voice.play_64_100 | voice.play_64_150 | voice.play_64_200)) | \
(wave.rec_11_16b & wave.play_11_16b) | \
(wave.rec_11_8b & wave.play_11_8b) | \
(oki.rec_24 & (oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | \
(oki.rec_32 & (oki.play_32_100 | oki.play_32_150 | oki.play_32_200)) | \
(ima.rec_24 & ima.play_24) | \
(ima.rec_32 & ima.play_32) | \
(gsm_ms.frgsm_rec & gsm_ms.frgsm_play) | \
g726.rec_32 | g726.play_32) )
Resource[0].DSPs = \
5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34\
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 61 62 63 64 65\
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94\
95
##########################################################################
#  The following resource definitions are REQUIRED in DLCP configurations
##########################################################################
Resource[1].Name                                = hdlc
Resource[1].Size                                = 48
Resource[1].TCPs                                = nocc
Resource[1].Definitions                         = ( hdlc.duplex )
Resource[1].DSPs                                = 2 3 4
#*************************************************************************************
# NOTE: DLMs to download
*************************************************************************************
DLMFiles[0]                                     = cg6500run
DebugMask                                       = 0x0
```

For more information about customizing the DLCP software, refer to *Integration overview* on page 43. For more information about NMS OAM, refer to the *NMS OAM System User's Manual*. Refer to the board's installation and developer's manual for more information about CG board keyword files.

6 Integrating the system

Integration overview

The Aztek protocol stacks use the NMS GR303 library and NMS V5 library to establish connections to and communicate with GR303 or V5.2 interfaces. The general rule for integrating different layers of a protocol is that the lower layers should start first and finish last. When integrating the NMS DLCP libraries (the lower layer) with Access303, Exchange303, and AV5 libraries (the upper layer), an application should always initialize the NMS DLCP libraries first, and tear them down last.

This section describes the steps you need to perform to integrate RDT, IDT, and AN applications. You can use the Aztek Integration Menu (AIM) tool with the *nms303tool* and *nmsv5tool* demonstration programs to test and troubleshoot the integration of the Aztek and NMS DLCP libraries.

Note: NMS recommends that you review this section thoroughly before designing RDT, IDT, or AN systems.

Integrating NMS GR303 and Aztek protocol software

This topic describes the steps for building integrated RDT or IDT applications that use the Aztek Access303 or Exchange303 protocol stack library and the NMS GR303 library.

- Specifying T1 link and HDLC channel locations
- Verifying T1 connections
- Testing the integration
- Building a complete GR303 application

Before integrating the various parts of the RDT or IDT system, create a mapping between the GR303 protocol structures and the data types defined by the layers of the Aztek GR303 stack and the NMS GR303 library. The following table shows mappings for data structures that are common for different functions within the Access303 and Exchange303 libraries and the NMS GR303 libraries:

GR303 protocol data	NMS GR303 data types	Access303 data types
Interface ID	NMS_GR303_INTERFACE_ID_T	AR303_INTERFACE_ID_T
Primary and standby DS1 locations	N/A	AR303_PATH_LOCATION_T
EOC or TMC protocol channel	N/A	AR303_PROTOCOL_CHANNEL_T
Primary and standby DS1 locations for an EOC or TMC protocol channel	N/A	AR303_CHANNEL_LOCATION_T
TMC and EOC channels for LAPD protocol to send and receive data	NMS_GR303_CHANNEL_LOCATION_T HDLC channel location with respect to hardware.	AR303_CHANNEL_T
TMC protocol: bearer-channel timeslot assignment for ports to make connections and process CAS signaling.	Logical timeslot number of a T1 trunk on a CG board.	AR303_CHANNEL_T
DS1 link	NMS_GR303_DS1_LOCATION_T	AR303_DS1_T
DS0 timeslot	Logical timeslot number of a trunk on a CG board.	AR303_DS0_T

Specifying T1 link and HDLC channel locations

To configure the GR303 library software to work with CG boards, applications must define a pair of data types to reference common location information for boards, trunks, and HDLC channels.

- The NMS_GR303_INTERFACE_ID_T data type specifies the ID associated with a particular lower level interface.
- The NMS_GR303_DS1_LOCATION_T data type provides information about the specific location of DS1 links. Applications use the following structure to specify the location of a particular DS1 link on a CG board:

```
struct {
DWORD boardNb;
DWORD trunkNb;
} CG;
```

where the data type specifies the following information:

Parameters	Description
boardNb	Logical board number of the board where the DS1 link resides (as defined by OAM).
trunkNb	Logical trunk number associated with the DS1 link (as defined by OAM).

- The NMS_GR303_CHANNEL_LOCATION_T data type provides information about the specific locations of HDLC channels. Applications use the following structure to specify the location of particular HDLC channels on a CG board:

```
struct {
DWORD boardNb;
DWORD trunkNb;
DWORD timeslotNb;
} CG;
```

where the data type specifies the following information:

Parameters	Description
boardNb	Logical board number of the associated board.
trunkNb	Logical trunk number on which the HDLC channel is located.
timeslotNb	Physical timeslot number to associate with the HDLC channel in the range 1 to 24. • Set the timeslot number to 12 for EOC data links • Set the timeslot number to 24 for TMC data links

Verifying T1 connections

You can use Aztek Access303 or Exchange303 software with NMS GR303 library software to develop GR303 compliant RDT-side applications. However, before you can run the application, connect the RDT side of the system to the IDT side of the GR303 protocol with a T1 cable. After you connect the T1 cable to the CG board, boot the CG board and start the application. Follow these steps to verify that the T1 links are synchronized:

Step	Action
1	Set up appropriate CT bus clocking for the system. In a single board system with T1 network connectivity, configure the board clocks to slave to a network interface. In a typical multiple-board system, configure one board to derive its clocking from the network and to act as the clock master for the other boards on the system. Configure the remaining boards to slave to the CT bus clock for inter-board synchronization. Refer to the *NMS OAM System User's Manual* or the appropriate board installation manual for more information about setting up CT bus clocking.
2	Boot the board and observe the trunk LEDs on the board's front panel. The LEDs remain briefly in an alarm condition after the board is booted, until frame synchronization is acquired. When the trunks leave the alarm state and became synchronized, the green LED remains lit for each trunk.
3	Run the *trunkmon* utility with the -b (board number) argument to monitor alarms and gather performance statistics for the T1 trunks. When all trunks are synchronized, *trunkmon* displays the alarm status for the board as NONE.
4	If the trunks connected to the external T1 cable link stay in an alarm state, connect a cross-over cable between any two trunks of the CG board. If the LEDs show that the trunks are in frame synchronization, this indicates that there is a problem with the T1 link rather than with the board.

For more information about CG board LEDs, the *trunkmon* utility, and using cross-over cables with CG boards, refer to the CG board documentation.

Testing the integration

Use the *nms303tool* and *aim303* programs to test the integration.

To use the *nms303tool* demonstration program:

Step	Action
1	Compile the program from the source code provided with the NMS DLCP software.
2	Start *nms303tool*.
3	Start the *aim303* utility (the AIM 303 task must be running).
4	Enter commands as needed from the *nms303tool* or *aim303* command line.

Refer to *Using the demonstration programs* on page 155 for more information about running *nms303tool.*

To use the *aim303* utility:

Step	Action
1	Compile the program from the source code provided with the Access 303 software.
2	Start the Access303 or Exchange303 software (by running *nms303tool*) and make sure that the Aztek library AIM task is set to listen on the appropriate socket for *aim303* messages.
3	Start the *aim303* utility by entering the following at the command line: `aim303 <hostname>` where ***hostname*** is the name of the machine running the GR303 application. The *aim303* program reports when the connection is established and displays a menu of available commands.
4	Enter commands as needed.

Refer to the *Aztek Integration Menu AIM-303 User Guide* for more information about using the *aim303* utility.

After you start the *nms303tool* and *aim303* programs, you can enter the commands at the command line of either program to perform operations on GR303 interfaces. These programs allow you to perform a variety of tasks, such as executing NMS GR303 or Aztek GR303 library functions, configuring and tracking task tracing information, verifying system status integration, and displaying status information about NMS GR303 interfaces.

Building a complete GR303 application

After completing the basic tasks required to integrate and verify the Aztek and NMS GR303 libraries, you can build an enhanced application by modifying the *nms303tool* source code to provide additional functionality or by creating a new application using functions from the NMS GR303 library.

Depending on the GR303 application's system requirements, you can use other NMS hardware and software to support specific functionality. For example, you can use the NMS Switching service to connect ports associated with a GR303 interface to specific CT bus timeslots, or use other NMS software to control robbed-bit signaling or media processing on connected ports.

For more information about the association between Aztek Access303 or Exchange303 library functions and NMS GR303 library functions when performing typical tasks, refer to *Developing GR303 applications* on page 51.

Integrating NMS V5 and Aztek AV5

This topic describes the steps for building an integrated AN application that uses the Aztek AV5 protocol stack library and the NMS V5 library.

- Specifying E1 link and HDLC channel locations
- Verifying E1 connections
- Testing the integration
- Building a complete AN application

Before integrating the various parts of the AN system, create a mapping between the V5.2 protocol structures and the data types defined by the layers of the AV5 protocol stack and by the NMS V5 library. The following table shows mappings for data structures that are common for different functions within the AV5 and NMS V5 libraries:

V5.2 protocol data	NMS V5 data types	Aztek AV5 data types
Interface ID	NMS_V5_INTERFACE_ID_T	V5_INTERFACE_ID_T
E1 link ID	N/A	V5_LE_LINK_ID_T
E1 location	NMS_V5_E1_LOCATION_T E1 trunk location on the CG board.	V5_E1_T (AV5 local ID)
Communication path (C-path) for Control, Link Control, PSTN, BCC, and Protection protocols	N/A	V5_C_PATH_T
Communication channel (C-channel) to carry a group of one or more C-paths	N/A	V5_LOGICAL_C_CHANNEL_T
Number of logical C-channels	num_channels function parameter when provisioning a variant.	V5_PROVISION_DATA_T (numLogCchan parameter)
Protection group number	N/A	V5_PROTECTION_GRP_NUM_T
Protection group standby channels	N/A	V5_PROTECT_STNDBY_T
	NMS_V5_CHANNEL_LOCATION_T HDLC channel location with respect to the hardware.	V5_PHYSICAL_C_CHANNEL_T for each standby
Provisioned variant	Capable of provisioning and destroying a standby variant.	V5_VARIANT_ID_T
PSTN or ISDN port on AN side	N/A	V5_AN_PORT_T
BCC protocol: Bearer-channel timeslot on E1	NMS_V5_E1_LOCATION_T and hardware-mapped logical timeslot number.	V5_E1_T and V5_TIME_SLOT_T
C-channels send/receive data	NMS_V5_CHANNEL_LOCATION_T HDLC channel location with respect to the hardware.	V5_E1_T and V5_C_CHANNEL_T

V5.2 protocol data	NMS V5 data types	Aztek AV5 data types
BCC protocol: Bearer-channel port location on AN side	Hardware-mapped logical timeslot number on an E1 trunk on an CG board.	V5_AN_PORT_T

Specifying E1 link and HDLC channel locations

To configure the NMS V5 library software to work with CG boards, applications must define a pair of data types to reference common location information for boards, trunks, and HDLC channels.

- The NMS_V5_INTERFACE_ID_T data type specifies the ID associated with a particular lower level interface.
- The NMS_V5_E1_LOCATION_T data type provides information about the specific location of E1 links. Applications use the following structure to specify the location of particular E1 trunks on a CG board:

```
struct {
DWORD boardNb;
DWORD trunkNb;
} CG;
```

where the data type specifies the following information:

Parameters	Description
boardNb	Logical board number of the board where the E1 link resides.
trunkNb	Logical trunk number associated with the E1 link.

- The NMS_V5_CHANNEL_LOCATION_T data type provides information about the specific locations of HDLC channels. Applications use the following structure to specify the location of particular HDLC channels on a CG board:

```
struct {
DWORD boardNb;
DWORD trunkNb;
DWORD timeslotNb;
} CG;
```

where the data type specifies the following information:

Parameters	Description
boardNb	Logical board number of the associated board.
trunkNb	Logical trunk number on which the HDLC channel is located.
timeslotNb	Physical timeslot number to associate with the HDLC channel. Specify timeslots 15, 16, and 31 as needed in accordance with the V5 protocol.

Verifying E1 connections

You can use Aztek AV5 library and NMS V5 library software to develop applications on the AN side of the V5.2 protocol. However, before you can run the application, connect the AN side of the system to the local exchange (LE) end of the V5.2 protocol through an E1 cable. After you connect the E1 cable to the CG board, boot the CG board, and start the LE side of the system. Follow these steps to verify that the E1 links are synchronized:

Step	Action
1	Set up appropriate CT bus clocking for the system. In a single board system with E1 network connectivity, configure the board clocks to slave to a network interface. In a typical multiple-board system, configure one board to derive its clocking from the network and act as the clock master for the other boards on the system. Configure the remaining boards to slave to the CT bus clock for inter-board synchronization. Refer to the *NMS OAM System User's Manual* or the appropriate board installation manual for more information about setting up CT bus clocking.
2	Boot the board and observe the trunk LEDs on the board's front panel. The LEDs remain briefly in an alarm condition after the board is booted, until frame synchronization is acquired. When the trunks leave the alarm state and are synchronized, the green LED remains lit for each trunk.
3	Run the *trunkmon* utility with the -b (board number) argument to monitor alarms and gather performance statistics for the E1 trunks. When all trunks are synchronized, *trunkmon* displays the alarm status for the board as NONE.
4	If the trunks connected to the external E1 link stay in an alarm state, connect a cross-over cable between any two trunks of the CG board. If the LEDs show that the trunks are in frame synchronization, this indicates that there is a problem with the E1 link rather than the board.

For more information about CG board LEDs, the *trunkmon* utility, and using cross-over cables with CG boards, refer to the CG board documentation.

Testing the integration

Use the *nmsv5tool* and *aimv5* programs to test the integration.

To use the *nmsv5tool* demonstration program:

Step	Action
1	Compile the program as an executable from the DLCP software source code.
2	Start *nmsv5tool*.
3	Start the *aimv5* utility (the AIM V5 task must be running).
4	Enter commands as needed from the *nmsv5tool* or *aimv5* command line.

For more information about using *nmsv5tool*, refer to *Using the demonstration programs* on page 155.

To use the *aimv5* utility:

Step	Action
1	Compile the program as an executable from the source code provided with the AV5 software.
2	Initialize the AV5 software (by running *nmsv5tool*) and make sure that the AV5 library AIM task is listening on the appropriate socket for the AIM messages.
3	Start *aimv5* by entering the following at the command line: `aimv5 <hostname>` where ***hostname*** is the name of the machine running the AN application. The *aimv5* program reports when the connection is established and displays a menu of available commands.
4	Enter commands as needed.

For more information about using *aimv5*, refer to the Aztek documentation.

After you start the *nmsv5tool* and *aimv5* programs, you can enter commands at the command line of either utility to perform operations on V5.2 interfaces. These programs allow you to perform a variety of tasks, such as executing API functions, toggling task tracing, verifying the integration status, and displaying status information.

Building a complete AN application

After completing the basic tasks to integrate and verify the AV5 and NMS V5 libraries, you can build an enhanced application by modifying the *nmsv5tool* source code to provide additional functionality or by creating a new application using functions from the NMS V5 library.

You can use the NMS Switching service to connect ports associated with a V5.2 interface to specific CT bus timeslots. Depending on the AN system requirements, you can also use other NMS software and hardware to provide access to PSTN or IP networks, or to provide digital loop carrier (DLC), hybrid fiber coax (HFC), or wireless local loop (WLL) support.

For more information about the association between Aztek AV5 library functions and NMS V5 library functions when performing typical tasks, refer to *Developing AN applications* on page 63.

7 NMS GR303 programming model

Developing GR303 applications

Use NMS hardware and software and NMS GR303 library functions to create remote digital terminal (RDT) or integrated digital terminal (IDT) applications that perform the following tasks:

- Configure T1 boards.
- Configure up to four HDLC controllers per GR303 interface.

  DS0 and DS1 locations for HDLC links are application-configurable, but for RDT applications must set these links to occupy timeslots 24 and 12 on two different T1 links.
- Perform automatic internal switching between CAS and RAW signaling modes when configuring HDLC channels.
- Send and receive HDLC data and transfer result reports.
- Provide buffering for outgoing and incoming messages on each HDLC data channel.
- Support ABCD robbed-bit signaling on a per timeslot basis.
- Support the switching fabric required for the TSI interface of the GR303 protocol.
- Provide library tracing capabilities.

HDLC device controllers perform the following tasks:

- Dynamically create and destroy HDLC instances.
- Initialize and reset HDLC instances.
- Activate and deactivate HDLC instances.
- Control the transmission and reception of HDLC frames.
- Perform statistics monitoring and configuration tasks.
- Control the transmission of asynchronous events and error reports.

Framer device controllers perform the following tasks:

- Monitor T1 links for alarms and errors.
- Control T1 links statistics queries and configuration.
- Configure signaling operation modes (CAS or RAW mode) on a per-timeslot basis.

The NMS GR303 library also provides physical layer access to the upper layers of the Aztek Access303 protocol stack. NMS GR303 library functions allow applications to perform the following tasks:

Tasks	Library functions
Managing the physical interface	• Initialize the software • Provision HDLC interfaces • Start and stop HDLC interfaces • Modify provisioned HDLC interfaces • Perform status and statistics queries
HDLC transmit and receive operations	• Send LAPD frames to HDLC channels • Receive LAPD frames from HDLC channels

Note: The NMS DLCP software is designed to work with the Aztek's Access303 or Exchange303 stack, but can also be integrated with third party GR303 stack software.

Using the NMS GR303 library

This topic presents the following information:

- Initializing the NMS GR303 library
- Provisioning an NMS GR303 interface
- Starting and monitoring an NMS GR303 interface
- Modifying an NMS GR303 interface
- Re-provisioning an NMS GR303 interface
- Sending and receiving HDLC channel data
- Retrieving channel and link status information
- Stopping an NMS GR303 interface
- Destroying an NMS GR303 interface
- Exiting the NMS GR303 library
- NMS GR303 library state model

Initializing the NMS GR303 library

When the GR303 system starts up, invoke **NMS_GR303Initialize** to load the NMS GR303 library, start the library's internal processing tasks, and initialize its internal software structures.

Invoke **NMS_GR303SetTrace** any time after loading the physical library. Arguments for **NMS_GR303SetTrace** specify the level of tracing performed by the library, and whether the library returns tracing information to the application or logs the information in a file.

Provisioning an NMS GR303 interface

After initializing the NMS GR303 library, call **NMS_GR303ProvisionInterface** to specify the hardware-specific HDLC channel locations, configure HDLC data processing function callbacks, and allocate memory for receiving data from the physical layer. The NMS GR303 library:

- Configures and resets HDLC channels on the board.
- Begins monitoring DS1 links associated with HDLC channels.
- Resets any internal statistics structures.
- Automatically sets the signaling mode to RAW for timeslots occupied by the HDLC channels.

Start the provisioned interface to enable data to flow through HDLC channels. The NMS GR303 library starts to monitor DS1 links as soon as the links are provisioned. Therefore, applications can call **NMS_GR303GetDS1Status** and **NMS_GR303ResetDS1Status** to obtain DS1 link status information or reset the status of existing DS1 links.

NMS recommends that applications provision and start the upper layers of the GR303 protocol after starting the NMS GR303 physical layer interface with **NMS_GR303StartInterface**. In some cases, the upper layers of the stack can expect incoming HDLC traffic after the interfaces are provisioned but before they are started.

GR303 applications must establish a mapping between the physical layer structures for hardware-specific HDLC device locations and upper layer GR303 protocol related structures. Because the physical layer is not aware of the protocol association for each hardware resource (for example, whether the DS1 link is active or standby mode, or whether a DS0 link is carrying EOC or TMC data) the application should map this information for the upper layers of the GR303 protocol stack.

Starting and monitoring an NMS GR303 interface

After provisioning the NMS GR303 interface, invoke **NMS_GR303StartInterface** to start each interface. When the application starts the interface, the HDLC channels provisioned on the interface become operational and data begins to flow through the associated T1 links. When the IDT on the local exchange side of the interface is connected and started, the application can expect LAPD data. Invoke the **NMS_GR303PhSendData** to transfer data from the upper layers of the GR303 stack to the physical layer. When the application invokes **NMS_GR303PhSendData**, the NMS GR303 library layer then sends the data to the specified HDLC channel.

Invoke **NMS_GR303GetDS1Status** and **NMS_GR303ResetDS1Status** to obtain or reset status information associated with DS1 links, and **NMS_GR303GetChannelStatistics** and **NMS_GR303ResetChannelStatistics** to access or reset statistical information associated with HDLC channels.

Modifying an NMS GR303 interface

The NMS GR303 library allows the application to add or modify HDLC channel locations on the interface at the physical layer without stopping and re-provisioning the interface. Use **NMS_GR303ModifyChannelLocation** to change the location of any provisioned HDLC channel or to add HDLC data links to a provisioned interface (when there are less than four existing provisioned channels).

Applications cannot remove existing channels from a provisioned interface. To reduce the number of HDLC channels at the physical layer, the application must destroy the existing interface, and then re-provision the interface with a smaller number of channels.

Re-provisioning an NMS GR303 interface

Perform the following steps to re-provision an interface:

Step	Action
1	Stop the interface (if it is started) with **NMS_GR303StopInterface**.
2	Destroy the current interface with **NMS_GR303DestroyInterface**.
3	Re-provision the interface with **NMS_GR303ProvisionInterface**.
4	Start the interface with **NMS_GR303StartInterface**.

Whenever possible, it is simpler to modify an interface with **NMS_GR303ModifyChannelLocation** than to destroy it and re-provision a new one.

Sending and receiving HDLC channel data

After starting an interface, invoke **NMS_GR303PhSendData** to asynchronously send data to a specific HDLC channel. The NMS GR303 library returns transfer request results to the application as soon as the data transfer is complete. If necessary, it also performs any buffering needed to transfer the data.

If the outgoing message queue overflows, the NMS GR303 library returns an EVENT_TX_QUEUE_FULL notification error. The library uses an application-defined callback function to transfer the HDLC error message data from the physical layer to the application. The library copies the data into memory that was allocated when the application provisioned the interface.

Retrieving channel and link status information

As soon as an application provisions an interface, the NMS GR303 library begins to monitor any DS1 links that carry provisioned HDLC channels, and to maintain channel statistics for the channels.

To query link status and channel statistic information, invoke **NMS_GR303GetChannelStatistics** and **NMS_GR303GetDS1Status**. **NMS_GR303GetDS1Status** returns current DS1 link alarm status information and error information collected over a specified period of time. **NMS_GR303GetChannelStatistics** reports success and error information for receive and transmit operations on a specified HDLC channel.

To reset DS1 link status information or HDLC channel statistics, invoke **NMS_GR303ResetChannelStatistics** or **NMS_GR303ResetDS1Status**.

Stopping an NMS GR303 interface

Call **NMS_GR303StopInterface** to stop an existing interface. When the interface stops, all HDLC data transfer through the interface ceases, and the interface remains provisioned but not operational.

The NMS GR303 library continues to monitor DS1 links on stopped interfaces, and the library does not automatically reset HDLC channel statistics for these links. Therefore, applications can still access or reset DS1 link statistics for stopped interfaces by invoking **NMS_GR303GetChannelStatistics** or **NMS_GR303ResetChannelStatistics**.

To restart a stopped interface, call **NMS_GR303StartInterface**.

Destroying an NMS GR303 interface

Invoke **NMS_GR303DestroyInterface** to destroy an existing interface. When an application destroys an interface, the HDLC flow of data ceases, provision information is destroyed, and the NMS GR303 library stops maintaining statistics information for any DS1 links and HDLC channels associated with the interface.

NMS_GR303DestroyInterface also resets the signaling mode of timeslots formerly occupied by the HDLC channels to CAS mode.

Exiting the NMS GR303 library

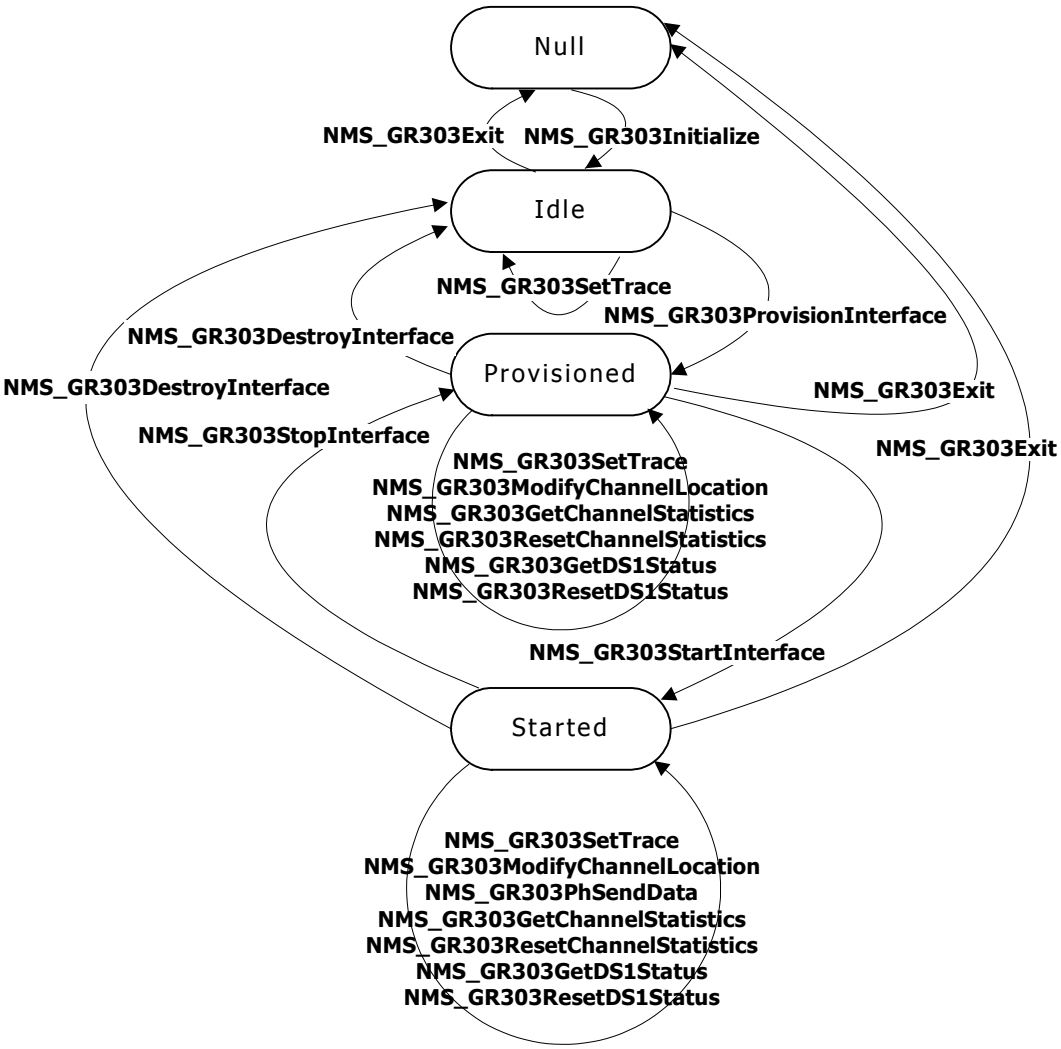
To exit the RDT system, call **NMS_GR303Exit**. When an application calls **NMS_GR303Exit**, the NMS GR303 library:

- Stops all provisioned HDLC and framer instances.
- Configures the signaling mode of any timeslots associated with HDLC channels back to CAS.
- Destroys all existing interfaces.

Exiting the GR303 application gracefully (that is, invoking **NMS_GR303StopInterface** and **NMS_GR303DestroyInterface** before invoking **NMS_GR303Exit**) brings the board to a known state and allows you to restart without rebooting the board.

NMS GR303 library state model

The following illustration shows state transitions for interfaces controlled by the NMS GR303 library. You must comply with the illustrated conditions when integrating NMS GR303 library functions into GR303 applications.



The following table describes the NMS GR303 library interface states:

State	Description
Null	The board is booted, configured, and initialized, but the NMS GR303 library is not loaded.
Idle	The NMS GR303 physical library is initialized. No communication link is established between the GR303 application and the boards. Library tracing can be configured.
Provisioned	The interface is provisioned on the physical layer. All necessary connections are made between the application and HDLC/framer instances on the boards. HDLC channels are configured, reset, and initialized, but disabled for transmit and receive operations. DS0 timeslots assigned to HDLC channels are internally set to RAW mode. All other DS0 timeslots remain in CAS mode. Applications can query the status of HDLC channels and DS1 links, modify provisioned interface configurations, and modify the tracing configuration.
Started	The HDLC channels are enabled for transmit and receive operations. Applications can query the status of HDLC channels and DS1 links, modify provisioned interface configurations, and modify the tracing configuration.

Programming scenarios

The following illustrations present scenarios for combining Aztek Access303 or Exchange303 library functions with NMS GR303 library functions to perform interface management and to conduct LAPD data communication over T1 connections. In these illustrations, the solid arrows represent commands and the dotted arrows represent data transfer.

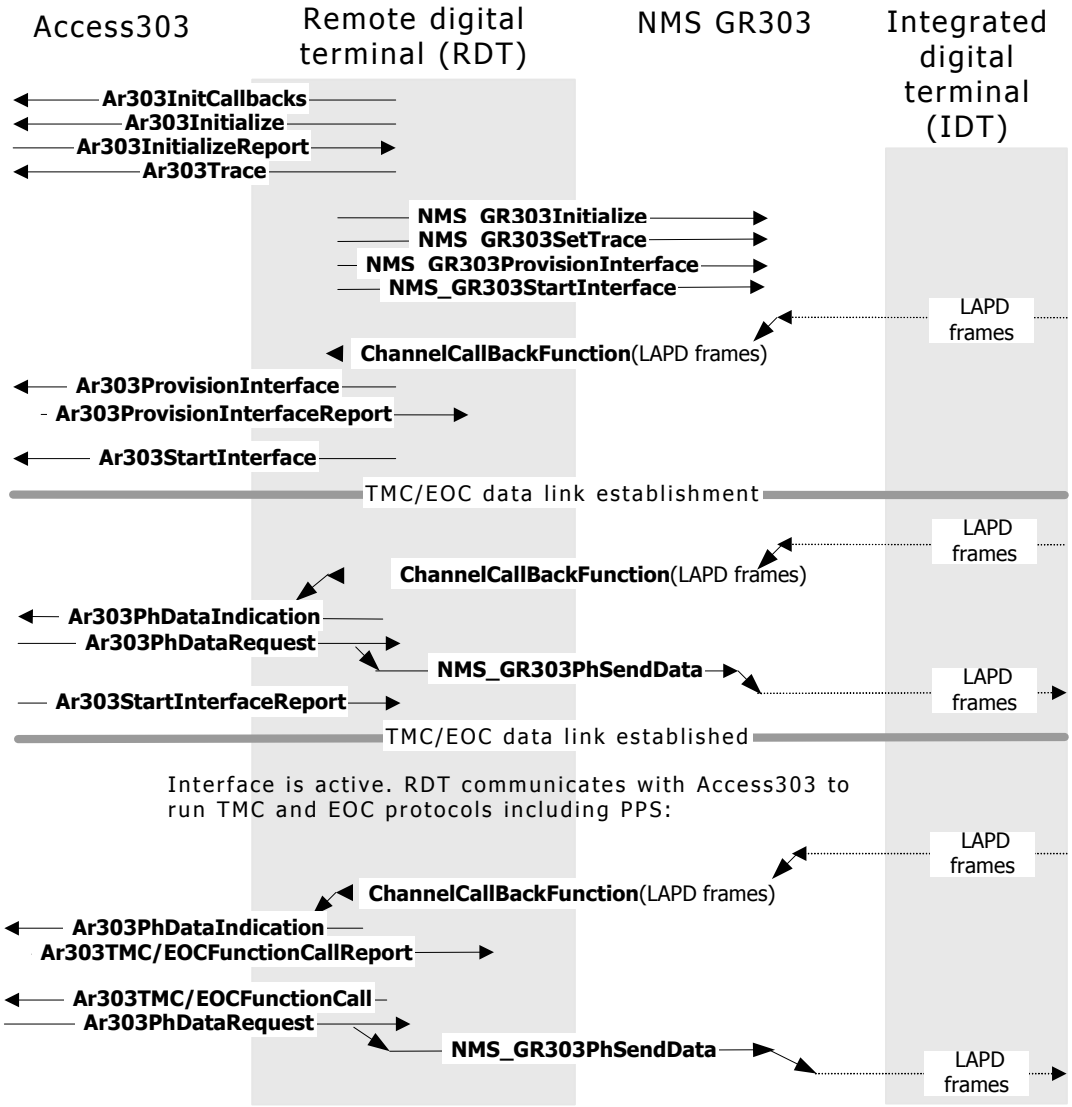
The following scenarios are presented:

- Initializing an interface
- Stopping and restarting without re-provisioning
- Stopping, re-provisioning, and starting
- Modifying standby locations

Refer to the Aztek Access303 or Exchange303 documentation more information about Aztek GR303 library functions.

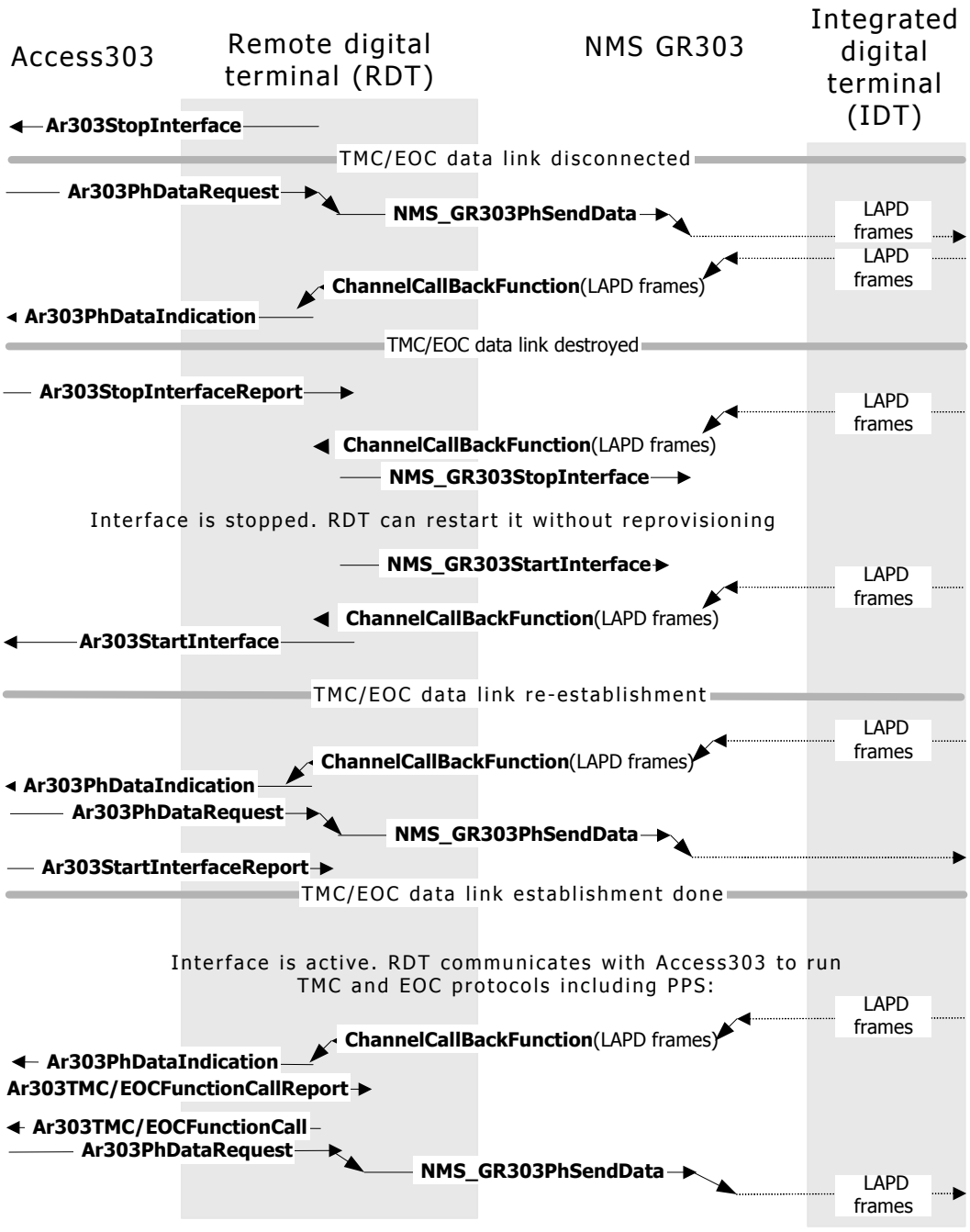
Initializing an interface

The following illustration shows functions used to initialize a GR303 interface for the RDT application:



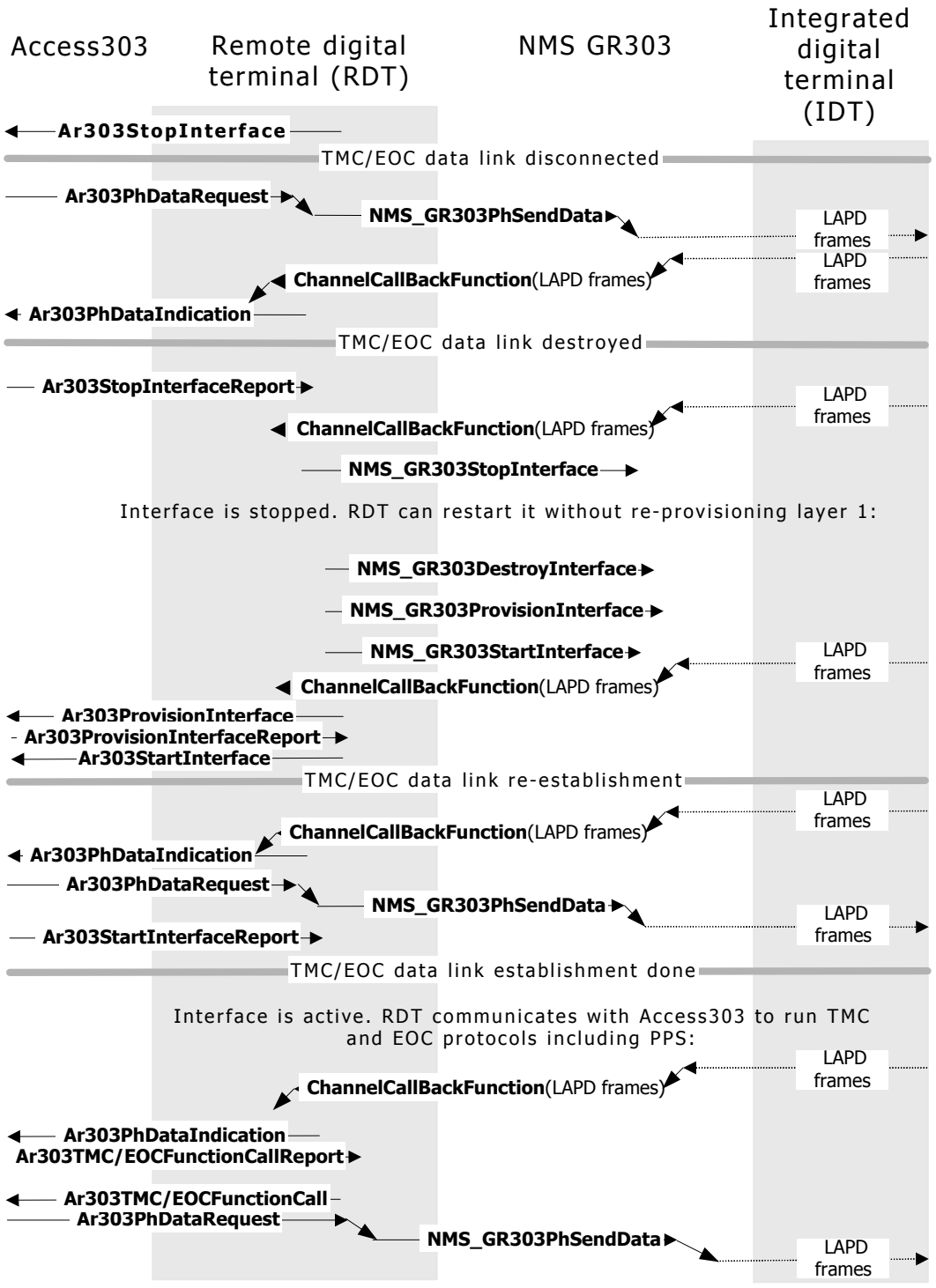
Stopping and restarting without re-provisioning

The following illustration shows functions used to stop and restart a GR303 interface without re-provisioning layer 1:



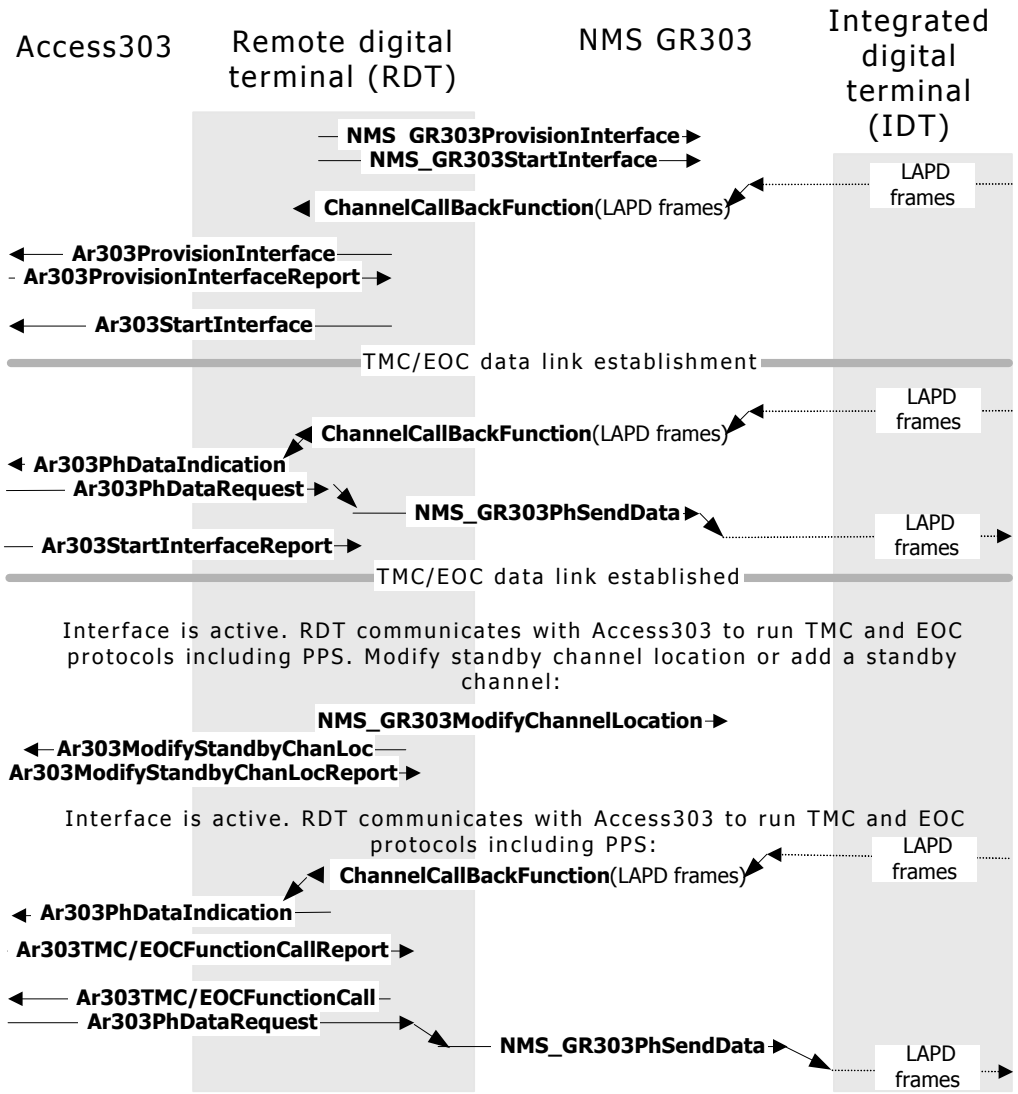
Stopping, re-provisioning, and starting

The following illustration shows functions used to stop a GR303 interface, re-provision all layers, and start the interface:



Modifying standby locations

The following illustration shows functions used to modify GR303 interface standby locations without stopping the interface:



8 NMS V5 programming model

Developing AN applications

Use NMS hardware and software with the NMS V5 library functions to create Access Network (AN) applications that perform the following tasks:

- Configure E1 boards.
- Configure up to three HDLC controllers per E1 link on the V5.2 interface.

  E1 link locations for HDLC links are application-configurable, but depending on the application requirements, AN applications must set these links to occupy timeslots 16, 15, or 31 on two different E1 links.
- Dynamically add and remove HDLC links required for ISDN over V5.2 support.
- Send and receive HDLC data and error reports.
- Buffer outgoing and incoming messages for each HDLC channel.
- Detect and report the following:
  - Loss of frame and loss of signal
  - Alarm indications (AIS) and remote alarm indications (RAIs)
  - Operations associated with the SA7 framing bit
  - Far end block error (FEBE) counters
- Support the lower level switching fabric required by the V5.2 protocol.
- Provide library tracing capabilities.

HDLC device controllers perform the following tasks:

- Dynamically configure, re-configure, and destroy HDLC instances.
- Initialize and reset HDLC instances.
- Activate and deactivate HDLC instances.
- Control the transmission and reception of HDLC frames.
- Perform statistics monitoring.
- Control the transmission of asynchronous events and error reports.

Framer device controllers perform the following tasks:

- Monitor E1 links for alarms and errors.
- Control E1 link statistic reports.
- Configure SA7 bit settings for E1 links.

The NMS V5 library also provides physical layer access to the upper layers of the Aztek AV5 protocol stack. NMS V5 library functions allow applications to perform the following tasks:

- Initialize the software.
- Provision interfaces (for active and standby variants).
- Start and stop interfaces.
- Switch to and from standby variants.
- Add and delete E1 links
- Monitor E1 links through link status and statistics queries.
- Perform HDLC transmit and receive operations such as sending and receiving LAPV frames on HDLC channels.
- Add and delete HDLC channels.

Note: The NMS DLCP software is designed to work with Aztek's AV5 software, but can also be integrated with other third party V5.2 implementation software.

Using the NMS V5 library

This topic presents the following information:

- Initializing the NMS V5 library
- Provisioning an NMS V5 interface
- Starting an NMS V5 interface
- Adding and deleting E1 links
- Adding and deleting HDLC channels
- Provisioning, switching to, and destroying a standby variant
- Sending and receiving HDLC channel data
- Monitoring E1 link status information
- Controlling SA7 bit values on E1 links
- Retrieving HDLC channel and E1 link status information
- Re-provisioning an existing interface
- Stopping an interface
- Destroying an interface
- Exiting the NMS V5 library
- NMS V5 library state model

Initializing the NMS V5 library

When the AN system starts up, invoke **NMS_V5Initialize**. to load the NMS V5 library, start the library's internal processing tasks, and initialize its internal software structures.

Invoke **NMS_V5SetTrace** any time after loading the NMS V5 library. Arguments for **NMS_V5SetTrace** specify the level of tracing performed by the library, and whether the library returns tracing information to the application or logs the information in a file.

Provisioning an NMS V5 interface

Before data can flow through HDLC channels and the NMS V5 library can return E1 status reports, applications must start V5.2 interfaces. After initializing the V5.2 physical layer software (the NMS V5 library), call **NMS_V5ProvisionInterface** to provision V5.2 interfaces. When invoking **NMS_V5ProvisionInterface:**

- Provide hardware-specific HDLC channel and E1 link location information.
- Define callback functions for processing HDLC data and E1 reports.
- Allocate memory for receiving data from the physical layer.

The NMS V5 library configures and resets the HDLC channels on the board, starts monitoring the provisioned E1 links, and resets its internal statistics structures. By default, the SA7 framing bit is set to 1 for newly provisioned E1 links.

The NMS V5 library begins to monitor E1 links after the application provisions the interfaces. Applications can then invoke **NMS_V5GetE1Status** and **NMS_V5ResetE1Status** to obtain or reset E1 link status information for the E1 links.

Note: NMS recommends that applications provision and start the upper layers of the V5.2 protocol stack after they start the physical layer with **NMS_V5StartInterface**. In some cases, the provisioned interfaces can expect incoming HDLC traffic before the application starts them.

Applications must establish a mapping between the physical layer HDLC channel and E1 link locations (which are NMS hardware-specific) and the upper layer V5.2 protocol structures. The physical layer does not track protocol information for the hardware resource assignments (for example, whether an E1 link is in active or standby mode, or what C-channel assignments apply to a particular timeslot). This information is out of scope of the physical layer and should be controlled by the upper layers of the V5.2 protocol software and by the application.

Starting an NMS V5 interface

To start NMS V5 interfaces, call **NMS_V5StartInterface** for each interface you want to start. After the application starts an interface, the HDLC channels provisioned on the interface become operational and data begins to flow across the E1 links. In addition, the NMS V5 library begins to report E1 alarms and remote SA7 framing bit value changes to the application.

If the local exchange is connected and active on the other side of the interface, the application can expect to receive LAPV messages though the E1 connections.

After starting the interface, you can invoke the following functions:

Function	Description
NMS_V5PhSendData	Passes data received from the upper layers of the V5.2 stack to the physical layer. The physical layer then sends the data to the appropriate HDLC channel.
NMS_V5GetE1Status	Retrieves information about an E1 link.
NMS_V5ResetE1Status	Resets the status of an E1 link.
NMS_V5GetChannelStatistics	Accesses HDLC channel statistics.
NMS_V5ResetChannelStatistics	Resets HDLC channel statistics.
NMS_V5AddE1	Adds an E1 link (associated with voice channels) to a provisioned active variant on a specified interface.
NMS_V5DeleteE1	Deletes an E1 link from an active variant of a provisioned interface.
NMS_V5AddChannel	Adds an HDLC channel (usually associated with an ISDN data channel) to a provisioned active variant on a specified interface.
NMS_V5DeleteChannel	Removes an HDLC channel from a provisioned active variant on a specified interface.

Adding and deleting E1 links

Applications can add or delete E1 links associated with voice and HDLC data without stopping and re-provisioning the active variant of a provisioned interface by calling **NMS_V5AddE1** or **NMS_V5DeleteE1**. If the application invokes **NMS_V5AddE1** for a started interface, the NMS V5 library automatically begins to monitor the new E1 link.

Later the application can add HDLC channels to an existing E1 link by calling **NMS_V5AddChannel**, or delete an HDLC channel from an existing (or already deleted E1 link) by calling **NMS_V5DeleteChannel**. The application cannot remove a provisioned E1 link if this link still has HDLC channels provisioned on it.

Adding and deleting HDLC channels

The NMS DLCP software provides NMS V5 library support for ISDN through the V5.2 protocol. Supporting ISDN on V5.2 interfaces requires that applications be able to dynamically add and remove ISDN channel locations to provisioned interfaces.

Access network (AN) applications use the following functions to dynamically add or delete HDLC channels that carry ISDN traffic over V5.2 interfaces:

Function	Description
NMS_V5AddChannel	Adds an HDLC link (usually associated with ISDN data channel over V5.2 interface) to an active provisioned interface variant.
NMS_V5DeleteChannel	Deletes an HDLC link (usually associated with ISDN data channel over V5.2 interface) from an active provisioned interface variant.

Call **NMS_V5AddChannel** or **NMS_V5DeleteChannel** when in the provisioned and started state as shown in the NMS V5 library state model.

Invoke **NMS_V5AddChannel** or **NMS_V5DeleteChannel** only for HDLC channels associated with a provisioned E1 link on a provisioned interface. When the application invokes **NMS_V5AddChannel** for a started interface, the NMS V5 library automatically starts the new HDLC link. After the new HDLC channel is added, the application can control the channel as if it had been provisioned with **NMS_V5ProvisionInterface**. Applications do not need to stop provisioned interfaces before invoking **NMS_V5DeleteChannel**.

Applications cannot use **NMS_V5AddChannel** or **NMS_V5DeleteChannel** to add or delete HDLC channels associated with the standby interface variants. To add or delete HDLC channels associated with the standby interface variants, invoke **NMS_V5DestroyStandByVariant** and then re-provision the interface with **NMS_V5ProvisionStandByVariant**.

Provisioning, switching to, and destroying a standby variant

Invoke **NMS_V5ProvisionStandByVariant** to configure a standby variant for any provisioned interface. HDLC channel and E1 link locations on standby variants can overlap with channel and link locations configured for active variants. The NMS V5 library saves the standby variant configuration but does not validate it.

To switch from the active NMS V5 interface to a standby variant, invoke **NMS_V5SwitchOverVariantData**. **NMS_V5SwitchOverVariantData** stops the active variant, destroys any associated provision information, and activates the provision for the standby variant. If the re-provisioning is successful, the standby variant becomes the provisioned active variant of the interface. However, the application must still call **NMS_V5StartInterface** to start the interface.

Note: Applications should call **NMS_V5SwitchOverVariantData** after the upper level of the V5.2 (in this case, the AV5 stack) interface has finished its switchover process.

To add or delete E1 links on a standby variant of an interface, you must first destroy the standby variant with **NMS_V5DestroyStandByVariant** and then re-provision the variant with **NMS_V5ProvisionStandByVariant**. If the call to **NMS_V5SwitchOverVariantData** fails, both the active and the failed standby variants of the existing interface are destroyed. The application must then invoke **NMS_V5DestroyInterface** and **NMS_V5ProvisionInterface** to re-provision the interface.

Sending and receiving HDLC channel data

Use **NMS_V5PhSendData** to asynchronously send data to HDLC channels on a started interface.

When the NMS V5 library finishes sending the data, it returns the results of the operation to the application. If necessary, the NMS V5 library performs internal buffering for outgoing messages associated with each HDLC data channel. If the outgoing message queue overflows, the NMS V5 library uses an application-defined callback function to return a notification error. The NMS V5 library uses a callback function to pass the incoming HDLC messages up to the application. The HDLC data is copied into memory that was allocated by the application when it provisioned the interface.

Monitoring E1 link status information

When the interface is provisioned, the NMS V5 library starts monitoring E1 links for alarms and remote SA7 framing bit changes. After the application starts an interface, the library invokes the application-defined callback function to report alarm events (LOS, LOF, AIS, RAI, normal frames, SA7 framing bit value updates). The first report on the current E1 state is sent to the application immediately when it starts. After that, the NMS V5 library only sends status when the E1 links change their state.

Use **NMS_V5GetE1Status** to query the NMS V5 library for E1 link state information any time after the interface has been provisioned. When the interface is destroyed, the NMS V5 library stops monitoring any E1 links associated with the interface. You can also use **NMS_V5ResetE1Status** to reset the status information associated with an E1 link.

Controlling SA7 bit values on E1 links

According to the link ID check procedure defined by the V5.2 protocol, the V5.2 protocol stack must be able to control the SA7 E1 framing bit. When an application provisions an interface, destroys an interface, or exits the NMS V5 library, the library automatically sets the SA7 bit to 1 on any associated E1 links. **NMS_V5SendSA7Bit** enables applications to set the SA7 bit value to 0 or 1 for a specified E1 link.

Retrieving HDLC channel and E1 link status information

When an application provisions an interface, the board automatically starts to monitor any provisioned E1 links. When the application starts the interface, the NMS V5 library begins to collect statistics for provisioned HDLC channels. Invoke **NMS_V5GetE1Status** to query E1 status information or invoke **NMS_V5GetChannelStatistics** to retrieve HDLC channel statistics for a particular HDLC channel.

NMS_V5GetE1Status returns current E1 link status and error history information (collected over an elapsed period of time). **NMS_V5GetChannelStatistics** reports success and error history information for both receive and transmit directions on a specified HDLC channel.

To reset the HDLC channel statistics or E1 link status, call **NMS_V5ResetChannelStatistics** or **NMS_V5ResetE1Status**.

Re-provisioning an existing interface

To re-provision an existing interface with NMS V5 library functions, perform the following steps:

Step	Action
1	Stop the interface (if the interface is started) with **NMS_V5StopInterface**.
2	Destroy the current interface with **NMS_V5DestroyInterface**. This function destroys both the active and standby variants of the interface. Destroyed E1 links have the SA7 framing bit set to 1.
3	Re-provision the interface by invoking **NMS_V5ProvisionInterface**. Provisioned E1 links have the SA7 framing bit set to 1.
4	Start the new interface with **NMS_V5StartInterface**.

Stopping an interface

Invoke **NMS_V5StopInterface** to stop an existing interface. When the application stops an interface, the interface remains provisioned but not operational. The flow of HDLC data through the interface stops and the NMS V5 library no longer returns E1 link alarm reports. However, the NMS V5 library continues to monitor E1 links and to maintain channel statistics.

Applications can still query E1 link status information and reset the status of E1 links associated with stopped interfaces. In addition, the NMS V5 library does not automatically reset HDLC channel statistics when the application stops an interface. The application can retrieve HDLC channel statistics or reset these statistics.

To restart a stopped interface, invoke **NMS_V5StartInterface**.

Destroying an interface

Invoke **NMS_V5DestroyInterface** to destroy an interface. When the interface is destroyed, the flow of HDLC data stops, and the NMS V5 library stops monitoring E1 links associated with the interface. The NMS V5 library destroys any provision information for the active and standby variants of the interface. Destroyed E1 links have SA7 framing bit set to 1.

To destroy only the standby variant of an existing interface, invoke **NMS_V5DestroyStandByVariant**.

Exiting the NMS V5 library

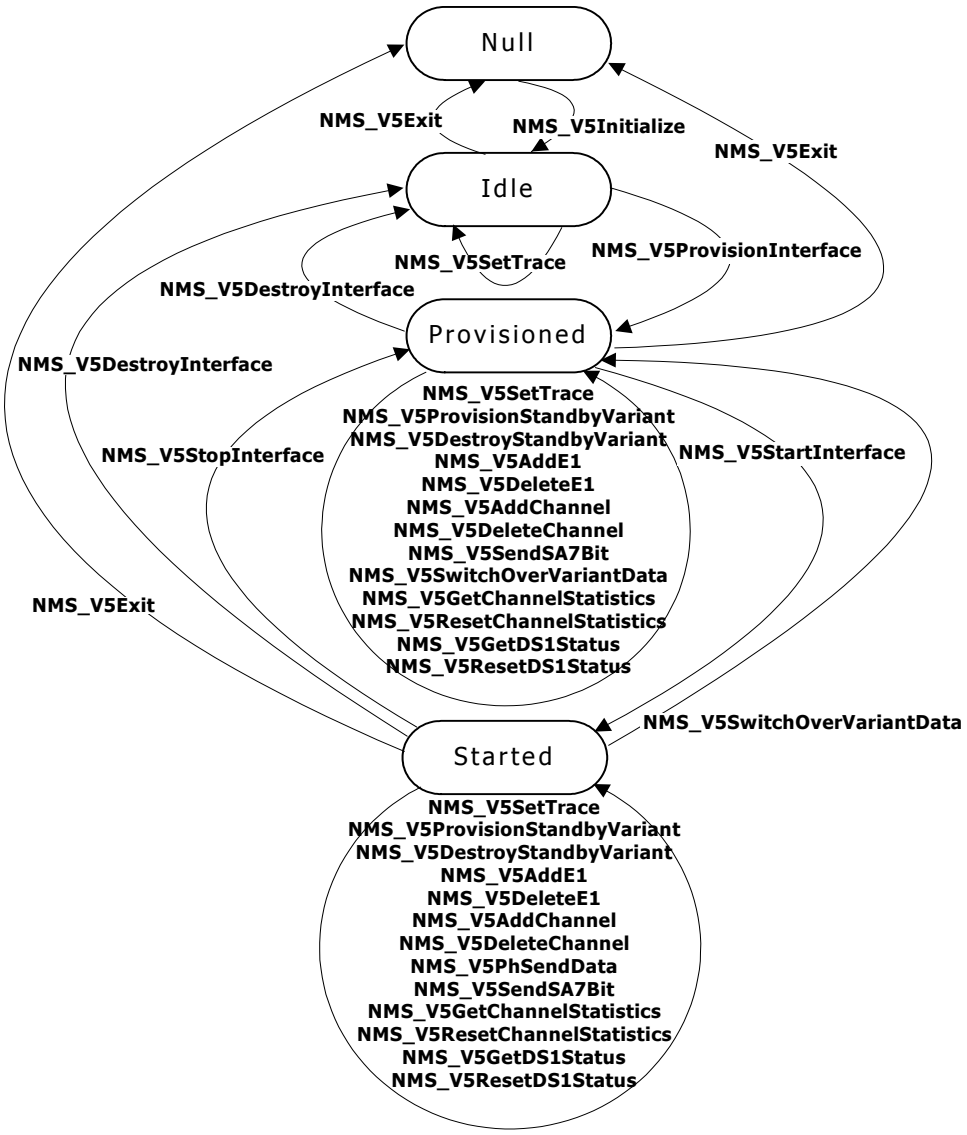
To exit the AN system, call **NMS_V5Exit**. When an application calls **NMS_V5Exit**, the NMS V5 library:

- Stops all provisioned HDLC channels.
- Disables E1 link monitoring.
- Resets the SA7 framing bit to 1 on all E1 links.
- Destroys all provisioned interfaces.

Exiting the AN application gracefully (that is, invoking **NMS_V5StopInterface** and **NMS_V5DestroyInterface** before invoking **NMS_V5Exit**) brings the boards to a known state, and allows you to restart the application without rebooting the boards.

NMS V5 library state model

The following illustration shows state transitions for interfaces controlled through the NMS V5 library. You must comply with the illustrated conditions when integrating NMS V5 library functions into AN applications.



The following table describes the NMS V5 library interface states:

State	Description
Null	The board is booted and initialized with the user configuration, but the NMS V5 physical library is not loaded.
Idle	The NMS V5 library is initialized. No communication link is established between the application and the boards. Library tracing can be configured.
Provisioned	The interface is provisioned on the physical layer. All necessary connections are made between the application and HDLC/framer instances on the boards. HDLC channels are configured, reset, and initialized, but disabled for receive and transmit operations. By default, the SA7 framing bit is set to 1 for provisioned E1 links. The application can query status of HDLC channels and DS1 links, modify provisioned configurations, and configure library tracing.
Started	The HDLC channels are enabled for transmit and receive operations. Applications can provision or destroy a standby variant, add or delete E1 links to an active interface variant, toggle the SA7 bit for provisioned E1 links, and query HDLC channel and E1 link status. Applications can also switch the interface to a standby variant and configure library tracing.

Programming scenarios

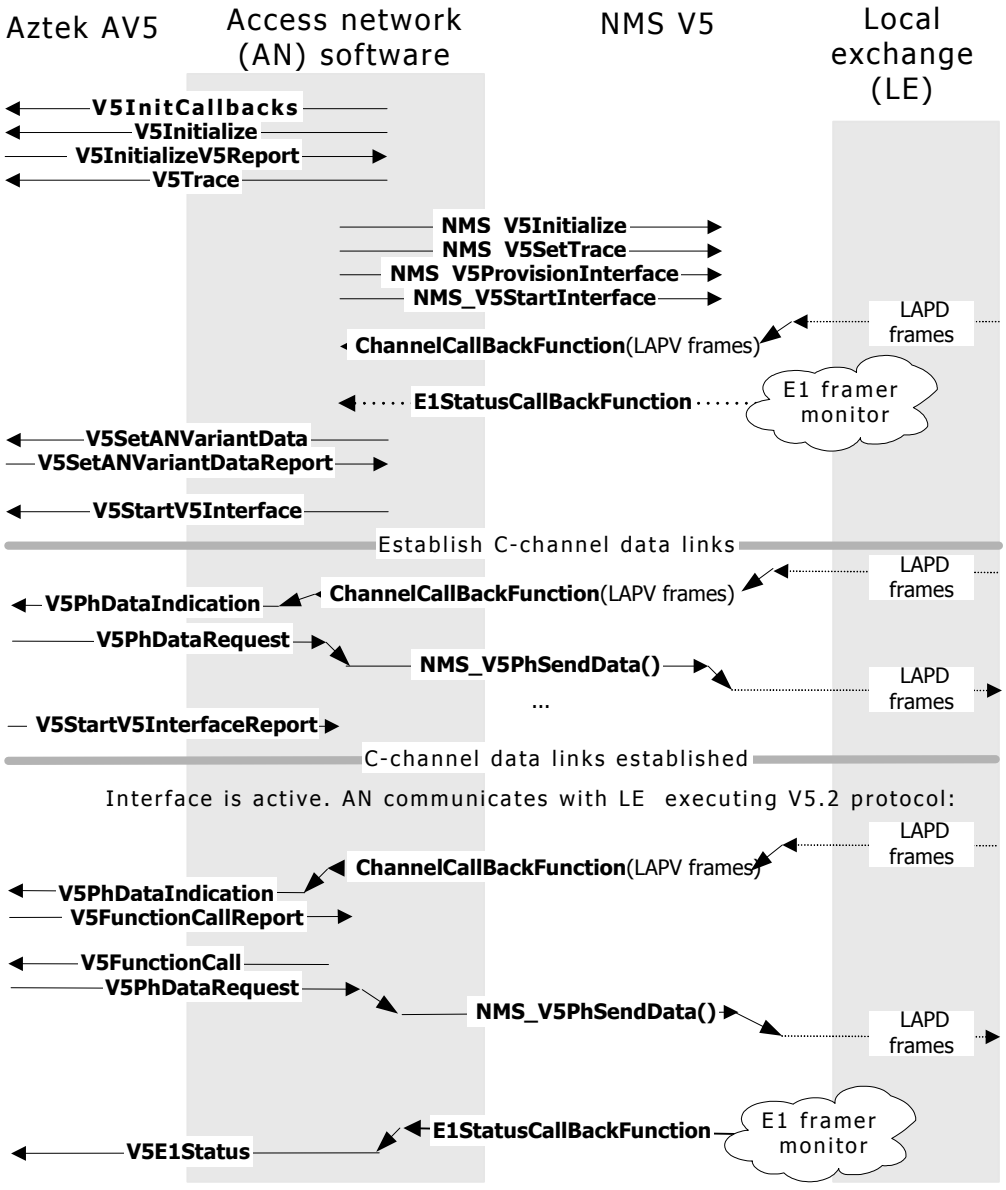
The following illustrations present scenarios for combining Aztek AV5 library and NMS V5 library function calls to perform interface management and LAPV data communication over E1 links. For more information about Aztek AV5 library functions, refer to the Aztek documentation.

The following scenarios are presented:

- Initializing an interface
- Stopping and restarting without re-provisioning
- Stopping, re-provisioning, and starting
- Checking link IDs
- Access network initiated switchover to a variant
- Local exchange initiated switchover to a variant

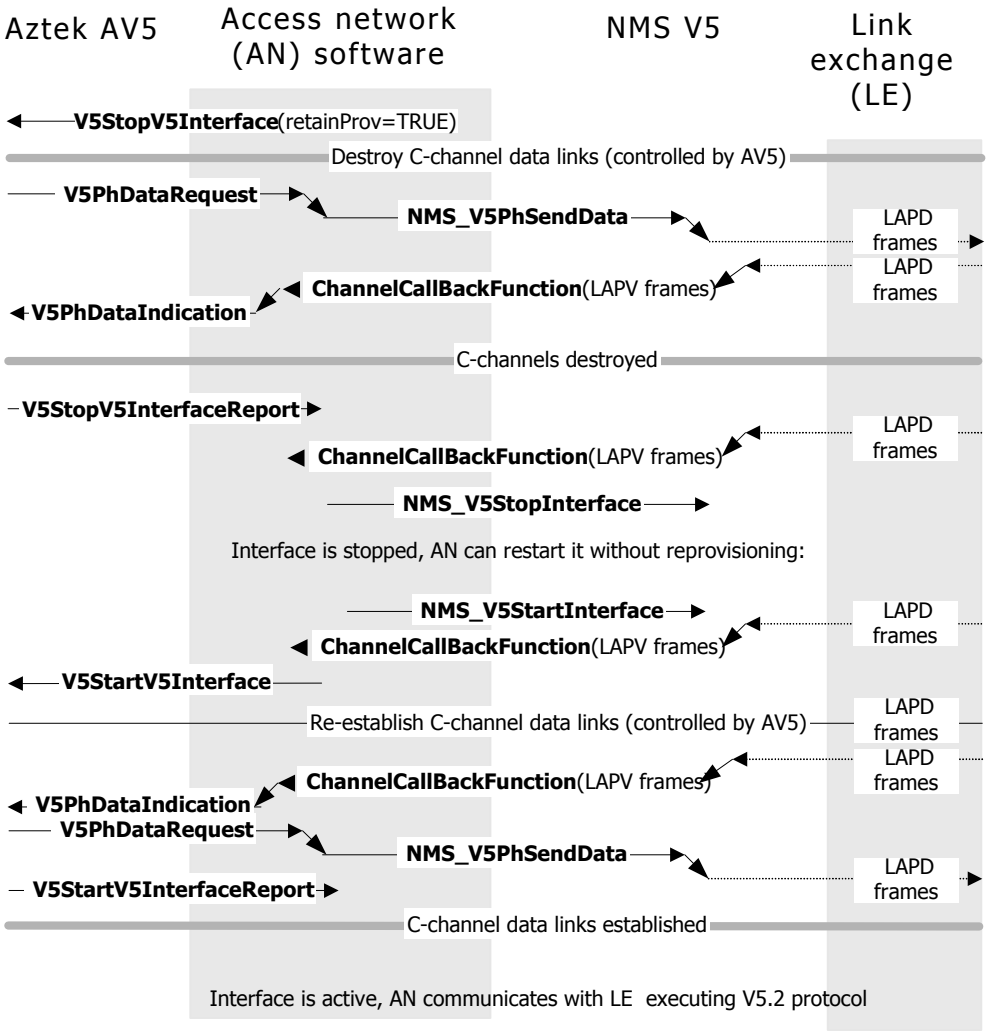
Initializing an interface

The following illustration shows functions used to initialize a V5.2 interface:



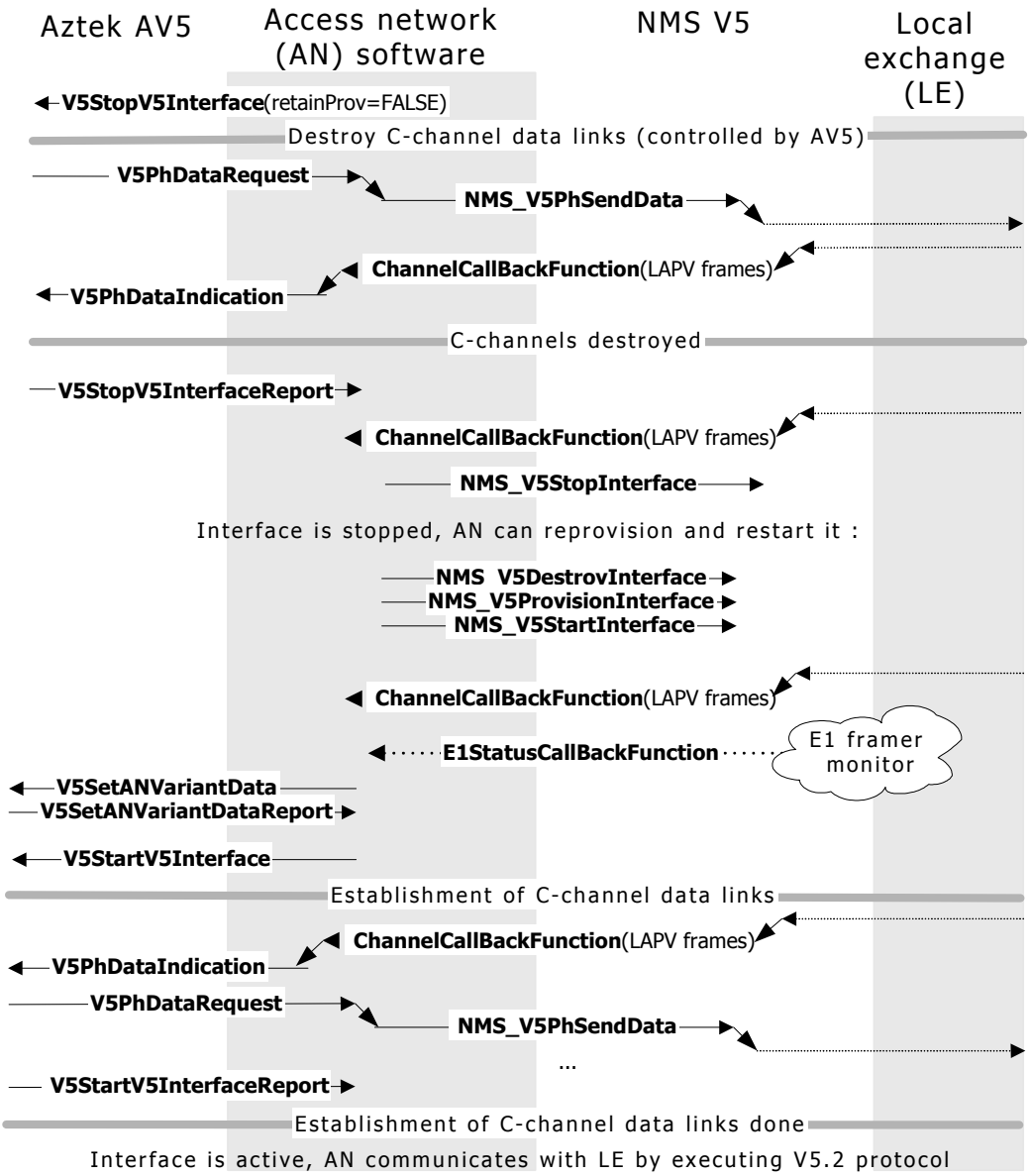
Stopping and restarting without re-provisioning

The following illustration shows functions used to the stop and restart a V5.2 interface without re-provisioning:



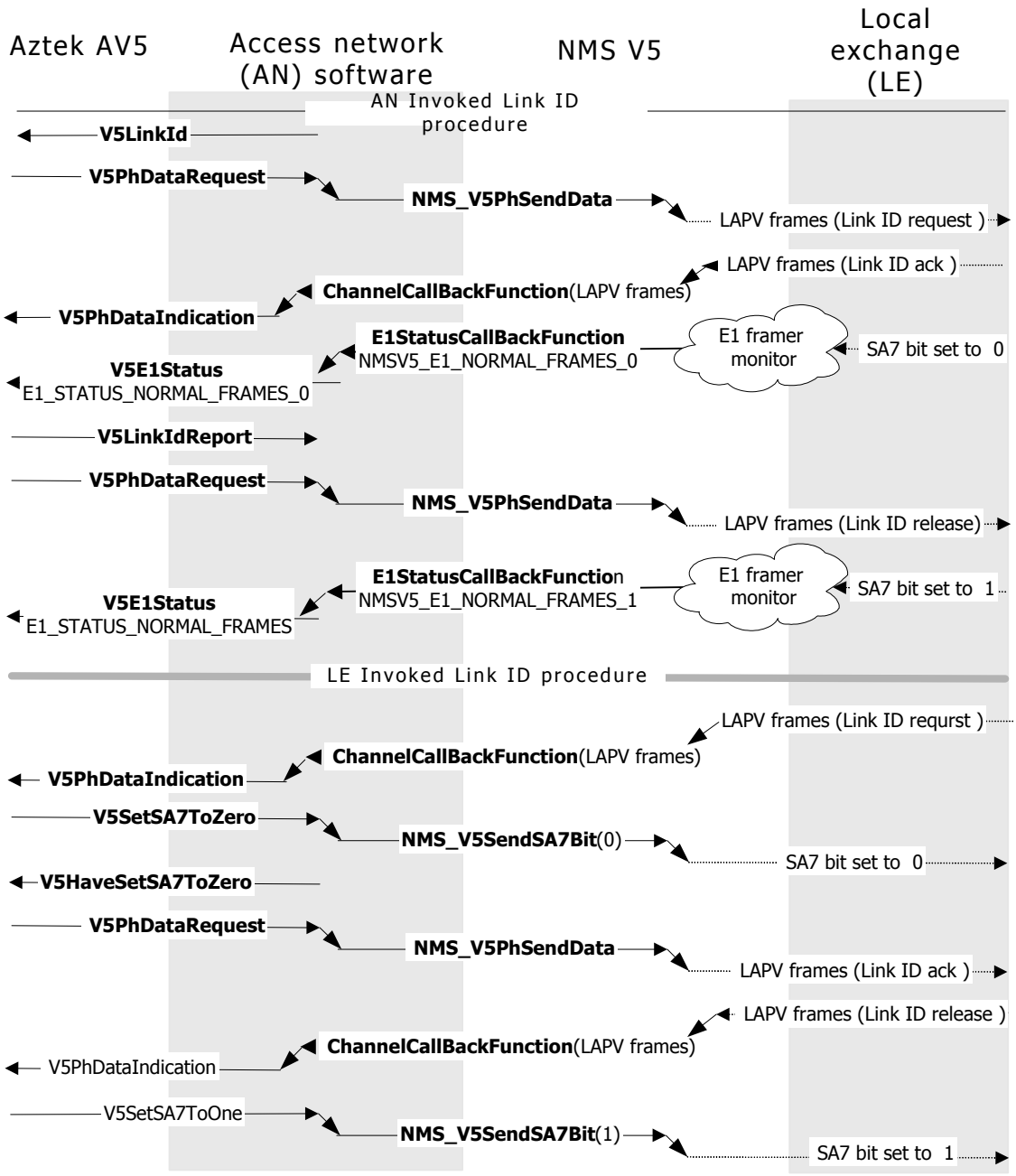
Stopping, re-provisioning, and starting

The following illustration shows functions used to stop, re-provision, and start a V5.2 interface:



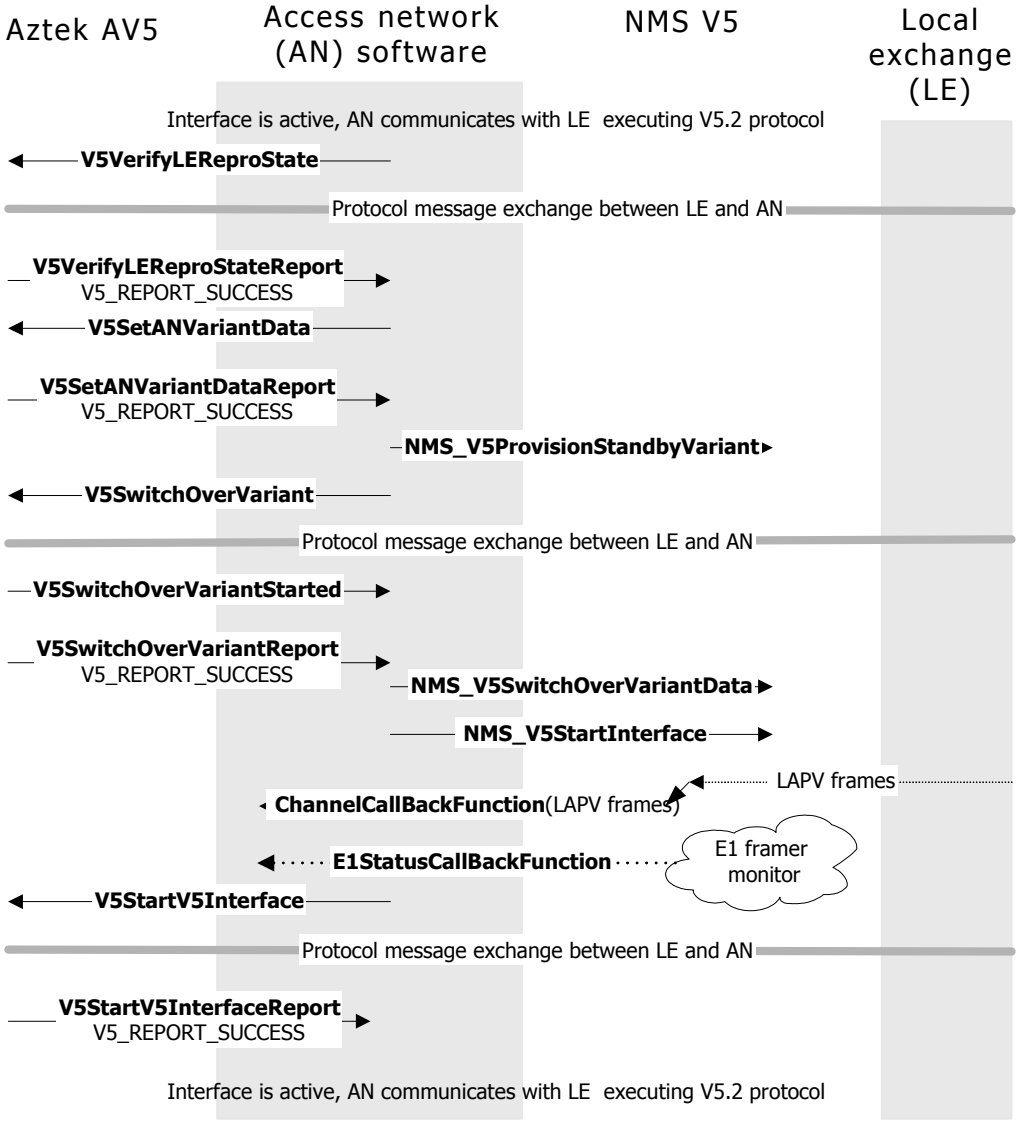
Checking link IDs

The following illustration shows functions used to check V5.2 link IDs:



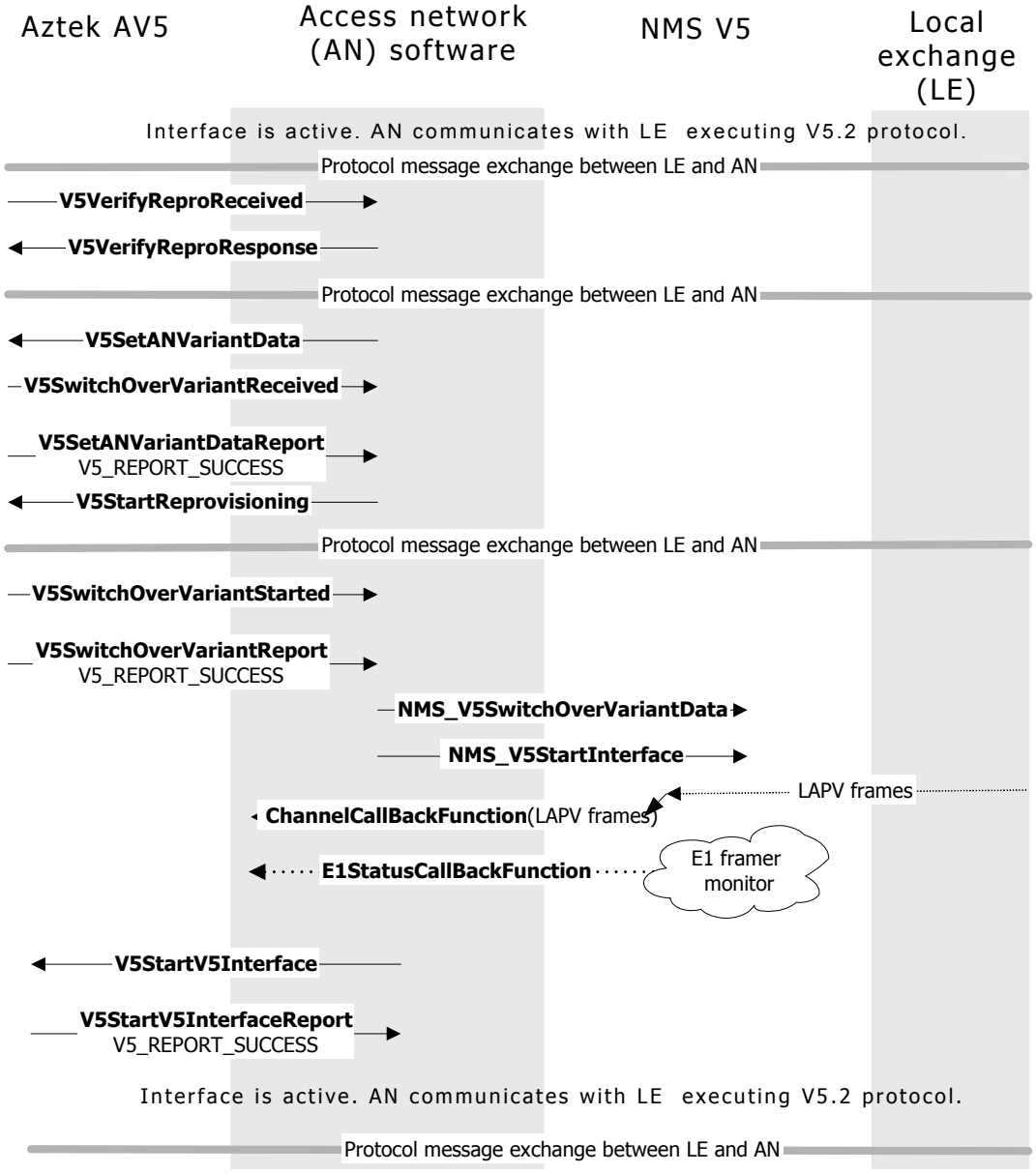
Access network initiated switchover to a variant

The following illustration shows functions used to provision a V5.2 variant interface and perform an access network initiated switchover operation:



Local exchange initiated switchover to a variant

The following illustration shows functions used to provision a V5.2 variant interface, and then to perform a local exchange initiated switchover operation:



9 NMS GR303 function reference

NMS GR303 library function summary

This topic summarizes NMS GR303 library functions according to task. All functions except **NMS_GR303PhSendData** are synchronous.

Initializing and exiting the NMS GR303 library functions

Use the following functions to initialize and exit the NMS GR303 library and to set trace levels for the library:

Function	Description
NMS_GR303Initialize	Loads and initializes the NMS GR303 library.
NMS_GR303SetTrace	Configures tracing for the NMS GR303 library.
NMS_GR303Exit	Performs an internal NMS GR303 library clean-up procedure and exits the library.

Creating, modifying, and destroying interfaces functions

Use the following functions to create and destroy physical layer interfaces with the NMS GR303 library:

Function	Description
NMS_GR303ProvisionInterface	Configures (provisions) the HDLC channels for the specified interface.
NMS_GR303ModifyChannelLocation	Moves a provisioned HDLC channel to a new location or adds a new HDLC channel to a specified interface without re-provisioning the entire interface.
NMS_GR303DestroyInterface	Destroys all connections for HDLC instances associated with a specified interface.

Controlling interfaces functions

Use the following functions to control physical layer interfaces with the NMS GR303 library:

Function	Description
NMS_GR303StartInterface	Starts the physical layer (HDLC channels) on a provisioned interface.
NMS_GR303StopInterface	Stops the physical layer (HDLC channels) of the provisioned interface.
NMS_GR303PhSendData	Sends a LAPD frame to a specified HDLC channel.

Retrieving and resetting DS1 link and channel information functions

Use the following functions to retrieve and reset link status information and channel statistics:

Function	Description
NMS_GR303GetDS1Status	Retrieves DS1 link status information for a specified DS1 link.
NMS_GR303ResetDS1Status	Resets DS1 link status information for a specified DS1 link.
NMS_GR303GetChannelStatistics	Retrieves HDLC channel statistics for a specified HDLC channel on an active provisioned interface.
NMS_GR303ResetChannelStatistics	Resets the HDLC channel statistics collected for a specified HDLC channel provisioned on the active interface variant.

Using the function reference

This section provides an alphabetical reference to the NMS GR303 functions. A prototype of each function is shown with the function description and details of all its arguments and return values. A typical function includes:

Prototype	The prototype is followed by a list of the function's arguments. NMS 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 defined. For more information, refer to the Data types overview.
Return values and **Events**	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. For more information about errors, refer to *NMS GR303 library errors* on page 173. For more information about events, refer to *NMS GR303 library channel callback events* on page 174.
Example	Example functions are excerpts taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

NMS_GR303DestroyInterface

Destroys all connections for HDLC instances associated with a specified interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303DestroyInterface** (
NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303DestroyInterface:

- Destroys the HDLC channels provisioned on the interface.
- Resets the signaling mode to CAS on timeslots occupied by disconnected HDLC channels.
- Clears the internal structures allocated for the specified interface.

Stop the interface with **NMS_GR303StopInterface** before calling **NMS_GR303DestroyInterface**, and call **NMS_GR303DestroyInterface** before re-provisioning the interface with **NMS_GR303ProvisionInterface**.

See also

NMS_GR303Exit

Example

```
void  DestroyInterface( void )
{
    NMS_GR303_RESULT_TNmsResult;
    DWORD            InterfaceId;

    printf("NMS_GR303DestroyInterface:\n");

    /* Get parameters */
    promptdw_nodft("Enter interfaceId", &InterfaceId);

    NmsResult = NMS_GR303DestroyInterface( InterfaceId );

    printf ("NMS_GR303DestroyInterface:Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303Exit

Performs an internal NMS GR303 library clean-up procedure and exits the library.

Prototype

NMS_GR303_RESULT_T **NMS_GR303Exit**()

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303Exit performs the following tasks:

- Destroys HDLC channels provisioned by the application (if they have not been destroyed) and resets the signaling mode to CAS for timeslots associated with provisioned HDLC channels.
- Flushes trace buffers, stops internal tasks, and frees memory allocated for the NMS GR303 library.

See also

NMS_GR303DestroyInterface, **NMS_GR303StopInterface**

Example

```
void  Exit( void )
{
     NMS_GR303_RESULT_T NmsResult;

       printf("NMS_GR303Exit:\n");

       NmsResult = NMS_GR303Exit();
    printf ("NMS_GR303Exit: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303GetChannelStatistics

Retrieves HDLC channel statistics for a specified HDLC channel on an active provisioned interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303GetChannelStatistics** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_CHANNEL_LOCATION_T ***channel_loc***, NMS_GR303_CHANNEL_STATISTICS_T ****channel_stat***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location for which you want to obtain the statistics.
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`  } CG;`
	`}NMS_GR303_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

channel_stat	Pointer to an application statistics buffer to receive the following structure for channel transmit and receive statistics:
	`typedef struct {`
	`  DWORD Size;  /* Size of structure  */`
	`  CHANNEL_STATISTICS_RECEIVE_T  chan_stat_rx;`
	`  CHANNEL_STATISTICS_TRANSMIT_T chan_stat_tx;`
	`} NMS_GR303_CHANNEL_STATISTICS_T;`
	
	`typedef struct {`
	`  DWORD Octets;`
	`  DWORD Frames;`
	`  DWORD Drops;`
	`  DWORD FifoOverruns;`
	`  DWORD Aborts;`
	`  DWORD CrcErrors;`
	`  DWORD NonAlignedOctets;`
	`  DWORD BufferOverflows;`
	`  DWORD MaxFrameLengthViolCnt1;`
	`} CHANNEL_STATISTICS_RECEIVE_T;`
	
	`typedef struct {`
	`  DWORD Octets;`
	`  DWORD Frames;`
	`  DWORD Drops;`
	`  DWORD FifoUnderruns;`
	`  DWORD FifoOverruns;`
	`} CHANNEL_STATISTICS_TRANSMIT_T;`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_CHANNEL	Specified HDLC channel is not provisioned on this interface.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_INVALID_PARMS	***channel_stat*** buffer pointer is NULL.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303GetChannelStatistics returns channel statistics for a specified HDLC channel within a NMS_GR303_CHANNEL_STATISTICS_T structure. Applications can only obtain statistics for HDLC channels that have been provisioned (with **NMS_GR303ProvisionInterface**) or added (with **NMS_GR303ModifyChannelLocation**).

The NMS_GR303_CHANNEL_LOCATION_T structure provides the following information:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	Physical timeslot number associated with the HDLC instance.

CHANNEL_STATISTICS_RECEIVE_T returns the following information:

Field	Description
Octets	Bytes received count.
Frames	Frames received count.
Drops	Frames dropped count (not supported on CG 6100C and CG 6500C boards).
FifoOverruns	FIFO overrun count (not supported on CG 6100C and CG 6500C boards).
Aborts	Receive abort count.
CrcErrors	Receive CRC errors count.
NonAlignedOctets	Non-aligned octets count (not supported on CG 6100C and CG 6500C boards).
BufferOverflows	Buffer overflows count (not supported on CG 6100C and CG 6500C boards).
MaxFrameLengthViolCnt1	Not supported.

CHANNEL_STATISTICS_TRANSMIT_T returns the following information:

Field	Description
Octets	Bytes transmitted count.
Frames	Frames transmitted count.
Drops	Frames dropped count.
FifoUnderruns	FIFO underrun count (not supported on CG 6100C and CG 6500C boards).
FifoOverruns	FIFO overrun count (not supported on CG 6100C and CG 6500C boards).

See also

NMS_GR303GetDS1Status, **NMS_GR303ResetChannelStatistics**

Example

```
void  GetChannelStatistics( void )
{
  NMS_GR303_RESULT_T              NmsResult;
  DWORD                           InterfaceId;
  NMS_GR303_CHANNEL_STATISTICS_T  Statistics;
  NMS_GR303_CHANNEL_LOCATION_T    ChannelLocation;



  printf("NMS_GR303GetChannelStatistics:\n");

  /* Get parameters */
  promptdw_nodft("Enter interfaceId", &InterfaceId);

  printf("Enter channel location:\n");
  GetChannelLocationNMS( &ChannelLocation );

  NmsResult = NMS_GR303GetChannelStatistics (InterfaceId,
                                        ChannelLocation,
                                        &Statistics);

  printf ("NMS_GR303GetChannelStatistics:
         Result=%s\n",PRINT_RESULT(NmsResult));

  if (NmsResult == NMSGR303_SUCCESS)
{
      printf("TxOctets     =%i\n", Statistics.chan_stat_tx.Octets);
      printf("TxFrames     =%i\n", Statistics.chan_stat_tx.Frames);
      printf("TxDrops      =%i\n", Statistics.chan_stat_tx.Drops);
      printf("TxUnderrun   =%i\n", Statistics.chan_stat_tx.FifoUnderruns);
      printf("TxOverrun    =%i\n", Statistics.chan_stat_tx.FifoOverruns);
      printf("RxOctets     =%i\n", Statistics.chan_stat_rx.Octets);
      printf("RxFrames     =%i\n", Statistics.chan_stat_rx.Frames);
      printf("RxDrops      =%i\n", Statistics.chan_stat_rx.Drops);
      printf("RxOverrun    =%i\n", Statistics.chan_stat_rx.FifoOverruns);
      printf("RxAbort      =%i\n", Statistics.chan_stat_rx.Aborts);
      printf("RxCrcError   =%i\n", Statistics.chan_stat_rx.CrcErrors);
      printf("RxNonAligned =%i\n", Statistics.chan_stat_rx.NonAlignedOctets);
      printf("RxBufferOverflow=%i\n",
                                  Statistics.chan_stat_rx.BufferOverflows);
  }

}
```

NMS_GR303GetDS1Status

Retrieves DS1 link status information collected for a specified DS1 link.

Prototype

NMS_GR303_RESULT_T **NMS_GR303GetDS1Status** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_DS1_LOCATION_T ***ds1_loc***, NMS_GR303_DS1_STATUS_T ****ds1_status***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

ds1_loc	DS1 link for which the NMS GR303 library returns status information. The DS1 location structure is defined as follows:
	`typedef union _NMS_GR303_DS1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD boardNb;`
	`  DWORD trunkNb;`
	`  } CG;`
	`}  NMS_GR303_DS1_LOCATION_T;`
	See the Details section for field descriptions.

ds1_status	Pointer to a buffer to receive the following status structure:
	`typedef struct {`
	`  DWORD Size;`
	`  NMS_GR303_DS1_STATUSMASK_T StatusMask;`
	`  DWORD Slips;`
	`  DWORD Es;`
	`  DWORD Ses;`
	`  DWORD Uas;`
	`  DWORD LineErrors;`
	`  DWORD FrameErrors;`
	`  DWORD ElaspedTime;`
	`}  NMS_GR303_DS1_STATUS_T;`
	See the Details section for field descriptions.

Return values

Return value	Description
SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_DS1	Specified DS1 link has no HDLC channels provisioned on it.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_INVALID_PARMS	Specified ***ds1_status*** buffer pointer is NULL.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303GetDS1Status returns status information associated with a specified DS1 link. For the NMS GR303 library to obtain the status information, the interface must be provisioned (with **NMS_GR303ProvisionInterface**) and the DS1 link must contain at least one HDLC channel that was configured on the specified interface. The application provides the following information in the NMS_GR303_DS1_LOCATION_T structure:

Field	Description
boardNb	Logical board number of the board where the DS1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the DS1 link (as defined by NMS OAM).

The GR303 library returns link status information in a NMS_GR303_DS1_STATUS_T structure that provides the following information:

Field	Description
Size	Size of the structure.
StatusMask	Current DS1 status bit mask. This can combine the following values: NMSGR303_DS1_LOS Loss of signal. NMSGR303_DS1_LOF Loss of frame. NMSGR303_DS1_AIS Alarm indication signal. NMSGR303_DS1_RAI Remote alarm indicator.
Slips	Number of slips.
Es	Number of errored seconds.
Ses	Number of severely errored seconds.
Uas	Number of unavailable seconds.
LineErrors	Number of line code violations.
FrameErrors	Number of frame bit errors and CRC errors.
ElaspedTime	Seconds since counters started, representing the duration of the observation time (in seconds). **Note:** All status fields provide counts of particular events that occur during the ElaspedTime interval.

The ElaspedTime parameter in the NMS_GR303_DS1_STATUS_T structure specifies the duration of the observation in seconds. All other status information represents counters of particular events. These events occur with an interval specified by ElaspedTime parameter.

Applications can use the & operator to extract bits in the StatusMask parameter (in the NMS_GR303_DS1_STATUS_T structure) to find out the current DS1 link condition. The following bit masks are defined:

Bit mask	Definition	Description
0x1	NMSGR303_DS1_LOS	Loss of signal.
0x2	NMSGR303_DS1_LOF	Loss of frame.
0x4	NMSGR303_DS1_AIS	Alarm indication signal.
0x8	NMSGR303_DS1_RAI	Remote alarm indication.

See also

NMS_GR303GetChannelStatistics, **NMS_GR303ResetDS1Status**

Example

```
void  GetDS1Status( void )
{
       NMS_GR303_RESULT_T         NmsResult;
       DWORD                      InterfaceId;
       NMS_GR303_DS1_LOCATION_T   DS1Location;
       NMS_GR303_DS1_STATUS_T     DS1Status = {0};

       printf("NMS_GR303GetDS1Status:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       printf("Enter DS1 location:\n");

       GetDS1LocationNMS( &DS1Location );
       NmsResult = NMS_GR303GetDS1Status(InterfaceId,
                                         DS1Location,
                                         &DS1Status);

       printf ("NMS_GR303GetDS1Status:
       Result=%s\n",PRINT_RESULT(NmsResult));

       if (NmsResult == NMSGR303_SUCCESS)
       {
        printf("StatusMask  LOS=%i, LOF=%i AIS=%i RAI=%i\n",
                  (DS1Status.StatusMask & NMSGR303_DS1_LOS) ? 1 : 0),
                  ((DS1Status.StatusMask & NMSGR303_DS1_LOF) ? 1 : 0),
                  ((DS1Status.StatusMask & NMSGR303_DS1_AIS) ? 1 : 0),
                  ((DS1Status.StatusMask & NMSGR303_DS1_RAI) ? 1 : 0));
        printf("Slips           =%i\n",   DS1Status.Slips);
        printf("Es              =%i\n",   DS1Status.Es);
        printf("Ses             =%i\n",   DS1Status.Ses);
        printf("Uas             =%i\n",   DS1Status.Uas);
        printf("LineErrors      =%i\n",   DS1Status.LineErrors);
        printf("FrameErrors     =%i\n",   DS1Status.FrameErrors);
        printf("ElaspedTime     =%i\n",   DS1Status.ElaspedTime);
       }
}
```

NMS_GR303Initialize

Loads and initializes the NMS GR303 library.

Prototype

NMS_GR303_RESULT_T **NMS_GR303Initialize** (NMS_GR303_BOARD_FAMILY_T ***board_family***, void ****init_parms***)

Argument	Description

board_family	Specifies the board family used on the system. The NMS_GR303_BOARD_FAMILY_T type is defined in the following structure:
	`typedef enum _NMS_GR303_BOARD_FAMILY_T`
	`{`
	`  NMSGR303_BOARD_FAMILY_CG = 0x01`
	`/* CG board family*/`
	`} NMS_GR303_BOARD_FAMILY_T;`

init_parms	Pointer to board-specific initialization parameters. For CG 6000 or CG 6000C boards, the application passes NULL. For CG 6100C and CG 6500C boards, the application passes a CTA context handle on which the Switching (SWI) service has been opened.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_ALREADY_INITIALIZED	NMS GR303 library is already initialized.
NMSGR303_FAILED_LOAD_LIB	Failed to load the library.
NMSGR303_INTERNAL_FAILURE	Failed to initialize internal structures, connections or processes.
NMSGR303_NO_MEMORY	Failed to allocate memory.

Details

NMS_GR303Initialize loads and initializes the NMS GR303 library. Applications invoke this function only once at start-up time, and before invoking any other NMS GR303 library functions.

If the system uses CG 6000 or CG 6000C boards, the application passes NULL as the ***init_parms*** argument.

When invoking **NMS_GR303Initialize** on systems that use CG 6100C or CG 6500C boards, applications use the ***init_parms*** argument to pass a pointer to a CTA context handle on which the Switching service is opened. When CG 6100C or CG 6500C boards reside in the system, applications must perform the following Natural Access operations before initializing the NMS GR303 library:

- Invoke **ctaInitialize** with the list of services including the Switching service.
- Invoke **ctaCreateQueue** to create an event queue.
- Invoke **ctaCreateContext** to create a context.
- Invoke **ctaOpenServices** to open appropriate Natural Access services, including the Switching service, on the context.

  ctaOpenServices returns a context handle that the application passes to the NMS GR303 library when invoking **NMS_GR303Initialize**.

If the application wants the NMS GR303 library to return trace information, it should invoke **NMS_GR303SetTrace** immediately after invoking **NMS_GR303Initialize**.

See also

NMS_GR303Exit, **NMS_GR303StartInterface**

Example

```
void  Initialize( void* InitParms )
{
      NMS_GR303_RESULT_T     NmsResult;

      printf("NMS_GR303Initialize:\n");

      NmsResult = NMS_GR303Initialize( g_BoardFamily, InitParms );
      printf ("NMS_GR303Initialize:
              Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303ModifyChannelLocation

Moves a provisioned HDLC channel to a new location or adds a new HDLC channel to a specified interface without re-provisioning the entire interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303ModifyChannelLocation** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_CHANNEL_LOCATION_T ****old_channel_loc***, NMS_GR303_CHANNEL_LOCATION_T ***new_channel_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

old_channel_loc	Pointer to an HDLC channel location structure for the channel to remove:
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`struct {`
	`DWORD boardNb;`
	`DWORD trunkNb;`
	`DWORD timeslotNb;`
	`} CG;`
	`} NMS_GR303_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.
	Set ***old_channel_loc*** to NULL if adding an HDLC channel.

new_channel_loc	HDLC channel location structure for the new channel location (see the ***old_channel_loc*** structure).

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_CHANNEL	Invalid channel location specified in ***old_channel_loc*** or ***new_channel_loc***.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.
NMSGR303_OUTOFRESOURCE	Maximum number of channels per GR303 interface (4 per interface) was reached.

Details

NMS_GR303ModifyChannelLocation modifies a channel location within an existing interface. If the application passes the ***old_channel_loc*** structure rather than NULL, **NMS_GR303ModifyChannelLocation** stops the specified HDLC instance (if the entire interface is started), re-configures the timeslots occupied by the HDLC channel back to CAS mode, and destroys the old HDLC channel.

If the application passes a NULL value for ***old_channel_loc*** value, the NMS GR303 library does not destroy any HDLC channels, but configures a new HDLC channel according to the information specified in the NMS_GR303_CHANNEL_LOCATION_T structure.

The NMS_GR303_CHANNEL_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	Physical timeslot number associated with the HDLC instance.

NMS_GR303ModifyChannelLocation creates, initializes, and resets the specified HDLC instance. If the specified interface has not been started, the channel is disabled by default. If the specified interface has been started, the channel is enabled by default.

The signaling mode of the timeslot occupied by the HDLC instance is set to RAW (or non-CAS). The maximum number of HDLC channels provisioned on an interface should not exceed four.

See also

NMS_GR303ProvisionInterface

Example

```
void  ModifyChannelLocation( void )
{
    NMS_GR303_RESULT_T            NmsResult;
    NMS_GR303_CHANNEL_LOCATION_T  NewChannelLoc, OldChannelLoc,
                                  *pOldChannelLoc;
    DWORD                         InterfaceId;
    char                          Selection;

    printf("NMS_GR303ModifyChannelLocation:\n");

    /* Get parameters */
    promptdw_nodft("Enter interfaceId", &InterfaceId);

    pOldChannelLoc = NULL;
    Selection = 'y';
    promptchar("Move provisioned channel ? (y/n)", &Selection );
    if(Selection == 'y')
    {
        printf("Enter 'Old' channel location:\n");
        GetChannelLocationNMS( &OldChannelLoc );
        pOldChannelLoc = &OldChannelLoc;
    }

    printf("Enter 'New' channel location:\n");

    NmsResult = NMS_GR303ModifyChannelLocation(InterfaceId,
                                               pOldChannelLoc,
                                               NewChannelLoc);

    printf ("NMS_GR303ModifyChannelLocation:
           Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303PhSendData

Sends a LAPD frame to a specified HDLC channel.

Prototype

NMS_GR303_RESULT_T **NMS_GR303PhSendData** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_CHANNEL_LOCATION_T ***channel_loc***, DWORD ***number_bytes***, void ****lapd_data***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location to which the NMS GR303 library sends the data.
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`} CG;`
	`}NMS_GR303_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

number_bytes	Size of the LAPD frame in bytes. The maximum number of bytes is 260.
lapd_data	Pointer to an application buffer containing the LAPD frame to send.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_CHANNEL	Specified HDLC channel is not provisioned on this interface.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_INVALID_PARMS	***lapd_data*** buffer pointer is NULL.
NMSGR303_INVALID_STATE	Interface is not started.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Events

Event	Description
NMSGR303_EVENT_TX_ERROR	Internal HDLC transmit error occurred. No buffer is attached to this error.
NMSGR303_EVENT_TX_FIFO_UNDERRUN	HDLC transmit FIFO underrun. No buffer is attached to this error. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_TX_FIFO_OVERRUN	HDLC transmit FIFO overrun. No buffer is attached to this error.

Event	Description
NMSGR303_EVENT_TX_QUEUE_FULL	The error indicates that the internal send queue is full and indicates an abnormal send rate. No buffer is attached to this error. Not supported for CG 6100C or CG 6500C boards.

Details

NMS_GR303PhSendData is asynchronous and returns without waiting for the board to report the send operation result. The specified interface must be started when the application calls **NMS_GR303PhSendData**.

The NMS_GR303_CHANNEL_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	Physical timeslot number associated with the HDLC instance.

The result of the send operation is reported later through the following application-defined callback structure:

```
typedef void(*NMS_GR303_CHANNEL_CALLBACK_T)
     (NMS_GR303_INTERFACE_ID_T interfaceId,
      NMS_GR303_CHANNEL_LOCATION_T channel_loc,
      NMS_GR303_CHANNEL_EVENT_T channel_event,
      void *databuffer,
      DWORD datasize);
```

The NMS GR303 library invokes the callback with the channel_event parameter set to a transmission result value. The transmission result is passed to the application, but the result does not indicate the send request with which it is associated.

See also

NMS_GR303StartInterface

Example

```
void SendData(BYTE  *FrameBuffer, DWORD NumberBytes )
{
    NMS_GR303_RESULT_T     NmsResult;
      DWORD                  InterfaceId;
     NMS_GR303_CHANNEL_LOCATION_T   ChannelLocation;

      printf("SendData:\n");

      /* Get parameters */
      promptdw_nodft("Enter interfaceId", &InterfaceId);

      printf("Enter channel location:\n");
      GetChannelLocationNMS( &ChannelLocation );

     NmsResult = NMS_GR303PhSendData( InterfaceId,
                                      ChannelLocation,
                                      NumberBytes,
                                      FrameBuffer );

     printf ("PhSendData: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303ProvisionInterface

Configures (provisions) the HDLC channels for the specified interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303ProvisionInterface** (
NMS_GR303_INTERFACE_ID_T ***interfaceId***, DWORD ***num_channels***,
NMS_GR303_CHANNEL_LOCATION_T ****channel_loc_array***,
NMS_GR303_CHANNEL_CALLBACK_T ****channel_callback***, void
****channel_rx_buffer***, DWORD ***channel_rx_buffersize***)

Argument	Description
interfaceId	Interface ID to associate with the provisioned interface. The valid range for interface ID values is 0 - 256.
num_channels	Number of HDLC channels to provision (the maximum is 4).

channel_loc_array	Pointer to an array of NMS_GR303_CHANNEL_LOCATION_T structures. The size of the array is specified by the ***num_channels*** argument. Each value in the ***channel_loc_array*** defines an HDLC location.
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`  } CG;`
	`} NMS_GR303_CHANNEL_LOCATION_T`
	The application must fill in a location structure that corresponds to the board family it is using (this is defined when invoking **NMS_GR303Initialize**).
	See the Details section for field descriptions.

channel_callback	Pointer to an application provided callback function. The library invokes this callback whenever it must pass an event associated with a provisioned HDLC channel. The application should return from this function as soon as possible. The callback function is defined as follows:
	`typedef void(*NMS_GR303_CHANNEL_CALLBACK_T)`
	`(NMS_GR303_INTERFACE_ID_T interfaceId,`
	`NMS_GR303_CHANNEL_LOCATION_T channel_loc,`
	`NMS_GR303_CHANNEL_EVENT_T channel_event,`
	`void *databuffer,`
	`DWORD datasize);`
	See the Details section for field descriptions.

channel_rx_buffer	Pointer to an application-allocated memory buffer.
channel_rx_buffersize	Size of the user allocated buffer specified by ***channel_rx_buffer***. The minimum requirement for ***channel_rx_buffersize*** is defined by the NMS_GR303_RX_BUFFER_SIZE parameter.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_ALREADY_INITIALIZED	Specified interface already exists.
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_PARMS	One or more of the function arguments are invalid.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303ProvisionInterface creates, initializes, and resets the specified HDLC instances, and leaves them configured but disabled. By default, **NMS_GR303ProvisionInterface** sets the signaling mode for the HDLC channel timeslots to RAW (or non-CAS).

Applications should call **NMS_GR303ProvisionInterface** before calling **NMS_GR303StartInterface**. After the application stops an interface with **NMS_GR303StopInterface**, the application does not need to re-provision the interface before restarting the interface (unless the application wants to change its configuration). To re-provision an existing configuration, the application must first call **NMS_GR303DestroyInterface** to destroy the provisioned configuration. However, the application can add or delete HDLC channels without re-provisioning the entire interface by invoking **NMS_GR303ModifyChannelLocation**.

Note: The application does not need to specify primary and backup HDLC channel assignments to NMS GR303.

The NMS_GR303_CHANNEL_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	Physical timeslot number associated with the HDLC instance.

When defining the **channel_callback** function, the application specifies the following arguments:

Argument	Description
interfaceId	The interface ID upon which the HDLC channel is provisioned.
channel_loc	HDLC location structure.
channel_event	HDLC channel event value, as defined in the next table.
databuffer	A pointer to the buffer attached to the event. When the **channel_callback** function returns, the library assumes that the application finished processing the channel_rx_buffer and that the buffer can be overwritten.
datasize	Size of the data saved in the buffer. The size should be 0 if there is no data attached to the event.

The channel_event field of the ***channel_callback*** function can include the following events:

Event name	Description
NMSGR303_EVENT_RX_ABORT	The HDLC channel received an ABORT sequence. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_BUFFER_OVERFLOW	An internal HDLC receive buffer overflow occurred. The HDLC received a frame of invalid length. No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_CRC_ERROR	The HDLC channel received a bad CRC for the receive frame. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_ERROR	An HDLC receive error was returned. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_FIFO_OVERRUN	A FIFO overrun occurred. No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_MAX_LENGTH_VIOLATION	Not supported.
NMSGR303_EVENT_RX_NON_ALIGNED_OCTET	The HDLC channel received an incomplete frame. No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_SUCCESS	A new HDLC frame was received and is available inside ***databuffer***. ***datasize*** is set to the frame size.
NMSGR303_EVENT_TX_ERROR	An internal HDLC transmit error occurred (no buffer is attached).
NMSGR303_EVENT_TX_FIFO_OVERRUN	An HDLC transmit FIFO overrun occurred. No buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_TX_FIFO_UNDERRUN	An HDLC transmit FIFO underrun occurred. No buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_TX_QUEUE_FULL	The HDLC transmit internal send queue is full. No buffer is attached. This indicates an abnormal send rate.

Example

```
void  ProvisionInterface( void )
{
    NMS_INTERFACE         NewInterface = {0};
    NMS_GR303_RESULT_T    NmsResult;
    char                  Selection;

    printf("NMS_GR303ProvisionInterface:\n");

    /* Set parameters */
    promptdw_nodft("Enter interfaceId", &NewInterface.InterfaceId);

    NewInterface.ChannelRxBufferSize = NMS_GR303_RX_BUFFER_SIZE;
    NewInterface.ChannelRxBuffer = calloc(
                                NewInterface.ChannelRxBufferSize, 1 );

    /* Configure channel parameters */
    do
    {
        Selection = 'y';
        promptchar("Add a new channel ? (y/n)", &Selection );
        if(Selection == 'y')
           {
            GetChannelLocationNMS(
            &NewInterface.NMS_Channels[NewInterface.NumChannels++] );
           }
    }
    while (Selection != 'n' && NewInterface.NumChannels <
                             NMS_GR303_CHANNEL_LOCATIONS);

    NmsResult = NMS_GR303ProvisionInterface (
                    NewInterface.InterfaceId,
                    NewInterface.NumChannels,
                    NewInterface.NMS_Channels,
                    ChannelCallBackFunction,
                    NewInterface.ChannelRxBuffer,
                    NewInterface.ChannelRxBufferSize);
    Result=%s\n",PRINT_RESULT(NmsResult));

    if( NMSGR303_SUCCESS != NmsResult )
    {
         /* Discard bad interface configuration */
         free(NewInterface.ChannelRxBuffer);
     }
}
```

NMS_GR303ResetChannelStatistics

Resets the HDLC channel statistics collected for a specified HDLC channel provisioned on an active interface variant.

Prototype

NMS_GR303_RESULT_T **NMS_GR303ResetChannelStatistics** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_CHANNEL_LOCATION_T ***channel_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC for which to reset statistics as specified in the following structure:
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`} CG;`
	`} NMS_GR303_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_CHANNEL	Specified HDLC channel is not provisioned on the interface.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303ResetChannelStatistics resets the channel statistics for the specified HDLC channel. The NMS_GR303_CHANNEL_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	Physical timeslot number associated with the HDLC instance.

See also

NMS_GR303GetChannelStatistics

Example

```
void  ResetChannelStatistics( void )
{
       NMS_GR303_RESULT_T            NmsResult;
       DWORD                         InterfaceId;
       NMS_GR303_CHANNEL_LOCATION_T  ChannelLocation;

       printf("NMS_GR303ResetChannelStatistics:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       printf("Enter channel location:\n");
       GetChannelLocationNMS( &ChannelLocation );

       NmsResult = NMS_GR303ResetChannelStatistics (InterfaceId,
                                                    ChannelLocation);

       printf ("NMS_GR303ResetChannelStatistics:
              Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303ResetDS1Status

Resets DS1 link status information collected for a specified DS1 link.

Prototype

NMS_GR303_RESULT_T **NMS_GR303ResetDS1Status** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_GR303_DS1_LOCATION_T ***ds1_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

ds1_loc	DS1 location for which to reset status information as specified in the following:
	`typedef union _NMS_GR303_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`  } CG;`
	`} NMS_GR303_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_DS1	Specified DS1 link has no HDLC channels provisioned on this interface.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303ResetDS1Status resets status information for a specified DS1 link. The NMS_GR303_CHANNEL_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number of the board where the DS1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the DS1 trunk (as defined by NMS OAM).

Applications can only request to reset status information for DS1 links that are associated with at least one HDLC channel.

See also

NMS_GR303GetDS1Status

Example

```
void  ResetDS1Status( void )
{
       NMS_GR303_RESULT_T        NmsResult;
       DWORD                     InterfaceId;
       NMS_GR303_DS1_LOCATION_T  DS1Location;

       printf("NMS_GR303ResetDS1Status:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       printf("Enter DS1 location:\n");
       GetDS1LocationNMS( &DS1Location );

       NmsResult = NMS_GR303ResetDS1Status( InterfaceId, DS1Location );

       printf ("NMS_GR303ResetDS1Status:
              Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303SetTrace

Configures NMS GR303 library tracing.

Prototype

NMS_GR303_RESULT_T **NMS_GR303SetTrace** (NMS_GR303_TRACEMASK_T ***trace_mask***, NMS_GR303_TRACE_CALLBACK_T ***user_trace_callback***, void ***user_trace_buffer***, DWORD ***user_trace_buffersize***, char ***user_file_name***)

Argument	Description
trace_mask	Trace mask controlling the information level reported to the application. Refer to the Details section for a list of trace masks that applications can specify.

user_trace_callback	Pointer to an application-defined callback function. Set this value to NULL if the library is saving tracing information to a file. The library uses this callback function when it has a trace buffer to pass to the application. The callback function is defined in the following way:
	`typedef void (*NMS_GR303_TRACE_CALLBACK_T)`
	`(char *tracebuffer,`
	`DWORD datasize);`
	See the Details section for field descriptions.

user_trace_buffer	Pointer to the application-allocated memory that the NMS GR303 library uses to pass trace data to the application.
user_trace_buffersize	Size of the application-allocated buffer specified by ***user_trace_buffer***. The minimum requirement for the ***user_trace_buffersize*** is set by NMS_GR303_MIN_TRACE_BUF_SIZE.
user_file_name	Pointer to the name of the log file if the NMS GR3030 library is logging trace information in a file. This value specifies the size of the trace blocks that the library uses to write to the file.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Failed to set up tracing.
NMSGR303_INVALID_PARMS	Invalid function arguments.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

Applications can use **NMS_GR303SetTrace** to configure the NMS GR303 library to return trace information through a defined callback function, or to log trace information into a specified file.

Applications can specify the following trace masks:

Trace mask	Description
NMSGR303_TRACEMASK_NONE	No tracing.
NMSGR303_TRACEMASK_FUNCTIONS	Log function calls and returns.
NMSGR303_TRACEMASK_ERRORS	Log HDLC transmit/receive errors.
NMSGR303_TRACEMASK_RX_BUF	Log received HDLC frames.
NMSGR303_TRACEMASK_TX_BUF	Log transmitted HDLC frames.
NMSGR303_TRACEMASK_HDLCERR	Log HDLC transmit/receive errors.

When defining a callback function, the application specifies the following information:

Field	Description
tracebuffer	A pointer to the beginning of the trace information inside the application-allocated memory.
datasize	The size of the data returned in the tracebuffer.

To initiate the callback mode, the application must pass a callback function pointer ***user_trace_callback***, allocate a buffer, and specify the buffer address (tracebuffer) and size (***user_trace_buffersize***) as **NMS_GR303SetTrace** arguments.

If the GR303 library receives a ***user_trace_callback*** value (and this value is not NULL), the library sets up the callback tracing mode and ignores the ***user_file_name*** value.

Note: After the **user_trace_callback** function returns, the library assumes that the application is finished using the data inside the tracebuffer, and that the library can reuse the memory. For this reason, applications must release **user_trace_callback** buffers as quickly as possible.

If the application decides to log trace information into a file, the application must set ***user_trace_callback*** and tracebuffer to NULL. If the application wants the library to perform internal buffering before logging tracing information in the file, the application must set the ***user_trace_buffersize*** to the trace block size. ***user_trace_buffersize*** specifies the size of an application-allocated buffer reserved for trace messages returned to the application. The size of ***user_trace_buffersize*** must either be greater than the value specified by the NMS_GR303_MIN_TRACE_BUFFER_SIZE parameter, equal to NMS_GR303_MIN_TRACE_BUFFER_SIZE, or zero.

If the application sets ***user_trace_buffersize*** to zero, the NMS GR303 library sends each trace message to the application immediately. However, even when the NMS GR303 library sends individual trace messages, the application must still reserve an amount of memory greater than or equal to the size of NMS_GR303_MIN_TRACE_BUFFER_SIZE.

The NMS GR303 library can return the following tracing messages with the NMSGR303_INTERNAL_FAILURE error:

Message	Hex	Description
PHYERR_INVALID_DEVICE	0x01	Board failed to define the HDLC or framer device type.
PHYERR_DEVICE_USED	0x02	Specified HDLC channel already exists in the NMS GR303 library.
PHYERR_INVALID_HANDLE	0x03	Invalid HDLC or framer handle passed. The board does not have an HDLC device configured with this handle.
PHYERR_BOARD_MESSAGE_FAILED	0x04	Failed to send a message to the board. Internal communication problem.
PHYERR_BOARD_RESPONSE_TIMEOUT	0x05	The board failed to respond to a message within a specified time interval.
PHYERR_INTERNAL_FAILURE	0x06	Failed to initialize or exit the NMS GR303 library.
PHYERR_INVALID_PARMS	0x07	Invalid input parameters.
PHYERR_NO_MEMORY	0x08	Failed to allocate memory.
PHYERR_ALREADY_INITIALIZED	0x09	NMS GR303 library has already been initialized.
PHYERR_NO_BUFFER_DEFINED	0x0A	No buffer provided for requested data.
PHYERR_OUTOFRESOURCE	0x0B	Cannot create more HDLC instances on the board.

Example

```
void  SetTrace( void )
{
    NMS_GR303_RESULT_T     NmsResult;
    NMS_GR303_TRACEMASK_T  TraceMask;
    BYTE                   FileName[64];
    char                   Selection;

    printf("NMS_GR303SetTrace:\n");

    prompthex("Enter TraceMask", &TraceMask);

    TraceBufferSize = NMS_GR303_MIN_TRACE_BUF_SIZE;

    /* Allocate buffer size not less than required minimum */
    g_pTraceBuffer = calloc(

    TraceBufferSize, 1);
    /* Set trace to a file */
    strcpy( FileName, "nms_gr303.log" );

    NmsResult =  NMS_GR303SetTrace(  TraceMask,
                                     NULL,
                                     NULL,
                                     TraceBufferSize,
                                     FileName);
    printf ("NMS_GR303SetTrace: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_GR303StartInterface

Starts the physical layer (HDLC channels) on a provisioned interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303StartInterface** (
NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_INVALID_STATE	Interface is already started.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303StartInterface starts a specified interface so that applications can use the interface. Applications must invoke **NMS_GR303Initialize** and **NMS_GR303ProvisionInterface** prior to calling **NMS_GR303StartInterface**.

When the HDLC channels on the interface are ready to send or receive frames, the NMS GR303 library invokes an application-defined callback (**channel_callback**) to send transmit and receive reports to the application.

See also

NMS_GR303StopInterface

Example

```
void  StartInterface( void )
{
       NMS_GR303_RESULT_T NmsResult;
       DWORD              InterfaceId;

       printf("NMS_GR303StartInterface:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       NmsResult = NMS_GR303StartInterface( InterfaceId );

       printf ("NMS_GR303StartInterface:
                  Result=%s\n",PRINT_RESULT(NmsResult));

}
```

NMS_GR303StopInterface

Stops the physical layer (HDLC channels) of the provisioned interface.

Prototype

NMS_GR303_RESULT_T **NMS_GR303StopInterface** (
NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSGR303_SUCCESS	
NMSGR303_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSGR303_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSGR303_INVALID_STATE	Interface is already stopped.
NMSGR303_NOT_INITIALIZED	NMS GR303 library was not initialized with **NMS_GR303Initialize**.

Details

NMS_GR303StopInterface disables HDLC channels on a specified interface and leaves the channels configured but inactive. When the NMS GR303 library finishes executing this function, it no longer sends receive and transmit channel events to the application. After an interface is stopped, the application can restart the interface at any time without re-provisioning it.

See also

NMS_GR303StartInterface, **NMS_GR303DestroyInterface**

Example

```
void  StopInterface( void )
{
       NMS_GR303_RESULT_T NmsResult;
       DWORD              InterfaceId;

       printf("NMS_GR303StopInterface:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       NmsResult = NMS_GR303StopInterface( InterfaceId );

       printf ("NMS_GR303StopInterface:
              Result=%s\n",PRINT_RESULT(NmsResult));
}
```

10 NMS V5 function reference

NMS V5 library function summary

This topic summarizes NMS V5 library functions according to task. All functions except **NMS_V5PhSendData** are synchronous.

Initializing and exiting the NMS V5 library functions

Use the following functions to initialize and exit the NMS V5 library and to set trace levels for the library:

Function	Description
NMS_V5Initialize	Loads and initializes the NMS V5 library.
NMS_V5SetTrace	Configures tracing for the NMS V5 library.
NMS_V5Exit	Performs an internal NMS V5 library clean-up procedure and exits the library.

Creating and destroying interfaces functions

Use the following functions to create and destroy primary and standby variant physical layer interfaces, and to switch from primary interfaces to standby variant interfaces:

Function	Description
NMS_V5ProvisionInterface	Configures (provisions) the HDLC channels for the specified interface.
NMS_V5ProvisionStandByVariant	Configures the HDLC channels and E1 links for an active variant on the specified interface.
NMS_V5DestroyInterface	Destroys all connections for HDLC instances associated with a specified interface.
NMS_V5DestroyStandByVariant	Destroys a provisioned standby variant on the specified interface.
NMS_V5SwitchOverVariantData	Switches the interface to a specified standby variant.
NMS_V5AddE1	Adds an E1 link associated with voice channels to a provisioned active variant on a specified interface.
NMS_V5DeleteE1	Deletes an E1 link from an active variant of a provisioned interface.
NMS_V5AddChannel	Adds an HDLC channel (usually associated with an ISDN data channel) to a provisioned active variant on a specified interface.
NMS_V5DeleteChannel	Removes an HDLC channel from a provisioned active variant on a specified interface.

Controlling interfaces functions

Use the following functions to control physical layer interfaces with the NMS V5 library:

Function	Description
NMS_V5StartInterface	Starts the physical layer (HDLC channels) of the provisioned interface.
NMS_V5StopInterface	Stops the physical layer of the provisioned interface.
NMS_V5PhSendData	Sends a LAPV frame to a specified HDLC channel (asynchronous).
NMS_V5SendSA7Bit	Sets the value of the SA7 E1 framing bit to 0 or 1.

Retrieving and resetting E1 link and channel information functions

Use the following functions to retrieve and reset link status information and channel statistics:

Function	Description
NMS_V5GetE1Status	Retrieves E1 link status information for a specified E1 link.
NMS_V5ResetE1Status	Resets E1 link status information for a specified E1 link.
NMS_V5GetChannelStatistics	Retrieves HDLC channel statistics for a specified HDLC channel on an active provisioned interface.
NMS_V5ResetChannelStatistics	Resets the HDLC channel statistics collected for a specified HDLC channel provisioned on the active interface variant.

Using the function reference

This section provides an alphabetical reference to the NMS V5 functions. A prototype of each function is shown with the function description and details of all its arguments and return values. A typical function includes:

Prototype	The prototype is followed by a list of the function's arguments. NMS 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 defined. For more information, refer to the Data types overview.
Return values and **Events**	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. For more information about errors, refer to *NMS V5 library errors* on page 176. For more information about events, refer to *NMS V5 library channel callback events* on page 178.
Example	Example functions are excerpts taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

NMS_V5AddChannel

Adds an HDLC channel (usually associated with an ISDN data channel) to a provisioned active variant of a specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5AddChannel** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_CHANNEL_LOCATION_T ***channel_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location specified by the following structure:
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD boardNb;`
	`  DWORD trunkNb;`
	`  DWORD timeslotNb;`
	`}  CG;`
	`} NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_CHANNEL	Specified HDLC channel is already provisioned on the specified interface.
NMSV5_INVALID_E1	An E1 link is not provisioned for this channel.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_PARMS	Invalid context handle was passed to **NMS_V5Initialize**.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.
NMSV5_OUTOFRESOURCE	Physical limit of HDLC channels on board is reached. No more HDLC channels can be added.

Details

NMS_V5AddChannel adds an HDLC link (usually associated with ISDN data over V5.2 interface) to an active variant of a provisioned interface without requiring the application to re-provision the interface. **NMS_V5AddChannel** creates, initializes, and resets the specified HDLC channel. When the NMS V5 library adds the channel, the library starts the channel if the interface is started, or leaves the channel configured but disabled if the interface is not started.

The NMS V5 library uses the call back function and the buffer the application defined when calling **NMS_V5ProvisionInterface** to pass HDLC data and events to the application. **NMS_V5AddChannel** performs any necessary internal switching between the DSP(s) processing the HDLC data and the channel's physical HDLC location on the E1 trunk. The E1 link with which the new HDLC channel is associated must be provisioned before the application can invoke **NMS_V5AddChannel** for that channel.

Caution:	Resetting the switch block on a board where HDLC channels have been provisioned automatically destroys the internal switching connections set up by the NMS V5 library between E1 interfaces and DSP cores.

The NMS_V5_CHANNEL_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC channel (as defined by NMS OAM).
TimeslotNb	Physical timeslot associated with the HDLC channel.

The NMS V5 library uses the **channel_callback** function (defined by the application when invoking **NMS_V5ProvisionInterface**) to pass HDLC frames and errors to the application.

See also

NMS_V5DeleteChannel, **NMS_V5AddE1**, **NMS_V5DeleteE1**

Example

```
void  AddChannel( void )
{
    NMS_V5_RESULT_T           NmsResult;
    DWORD                     InterfaceId;
    NMS_V5_CHANNEL_LOCATION_T NMSNewChannelLoc;

    printf("NMS_V5AddChannel:\n");

    /* Get parameters */
    promptdw_nodft("Enter interfaceId", &InterfaceId);
    GetChannelLocationNMS( &NMSNewChannelLoc );

    NmsResult = NMS_V5AddChannel( InterfaceId, NMSNewChannelLoc );

    printf ("NMS_V5AddChannel: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5AddE1

Adds an E1 link associated with voice channels to a provisioned active variant on a specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5AddE1** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_E1_LOCATION_T ***e1_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

e1_loc	E1 link location specified by the following structure:
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`} CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMS_INVALID_E1	Specified E1 link is already provisioned.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5AddE1 adds an E1 link associated with voice or HDLC channels to an active variant of a provisioned interface without requiring the application to re-provision the interface. When the application invokes **NMS_V5AddE1**, the NMS V5 library begins monitoring status information associated with the E1 link. If the application starts the interface, the NMS V5 library reports alarms and SA7 bit changes to the application through application-configured callbacks. Applications cannot call **NMS_V5AddE1** for standby variants.

The NMS_V5_E1_LOCATION_T structure includes the following information:

Field	Description
boardNb	Logical board number where the E1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the E1 link (as defined by NMS OAM).

See also

NMS_V5DeleteE1

Example

```
void  AddE1( void )
{
       NMS_V5_RESULT_T        NmsResult;
       DWORD                  InterfaceId;
       NMS_V5_E1_LOCATION_T   NMSNewE1Loc;

       printf("NMS_V5AddE1:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       GetE1LocationNMS( &NMSNewE1Loc );

       NmsResult = NMS_V5AddE1( InterfaceId, NMSNewE1Loc );
}
```

NMS_V5DeleteChannel

Removes an HDLC channel from a provisioned active variant on a specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5DeleteChannel** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_CHANNEL_LOCATION_T ***channel_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location specified by the following structure:
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD boardNb;`
	`  DWORD trunkNb;`
	`  DWORD timeslotNb;`
	`}  CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_CHANNEL	Specified HDLC channel is not provisioned on the specified interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5DeleteChannel removes an HDLC link (usually associated with ISDN data over V5.2 interface) from an active variant of a provisioned interface without requiring the application to re-provision the interface. **NMS_V5DeleteChannel** stops and destroys the specified HDLC channel and destroys the switch connections made for this channel between an E1 channel location and a DSP processing HDLC data. **NMS_V5DeleteChannel** does not affect the operation of other HDLC channels and E1 links provisioned on this interface.

The NMS_V5_CHANNEL_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC channel (as defined by NMS OAM).
TimeslotNb	Physical timeslot associated with the HDLC channel.

See also

NMS_V5AddChannel, **NMS_V5AddE1**, **NMS_V5DeleteE1**

Example

```
void  DeleteChannel( void )
{
    NMS_V5_RESULT_T           NmsResult;
    DWORD                     InterfaceId;
    NMS_V5_CHANNEL_LOCATION_T NMSChannelLoc;

    printf("NMS_V5DeleteChannel:\n");

    /* Get parameters */
    promptdw_nodft("Enter interfaceId", &InterfaceId);

    /* Ask for channel location parameters */
    GetChannelLocationNMS( &NMSChannelLoc );

    NmsResult = NMS_V5DeleteChannel( InterfaceId, NMSChannelLoc );

    printf ("NMS_V5DeleteChannel: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5DeleteE1

Deletes an E1 link from an active variant of a provisioned interface.

Prototype

NMS_V5_RESULT_T **NMS_V5DeleteE1**(NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_E1_LOCATION_T ***e1_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

e1_loc	E1 link location specified by the following structure:
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`}  CG;`
	`}NMS_V5_E1_LOCATION_T;`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5DeleteE1 deletes an E1 link associated with voice or HDLC channels from an active variant of a provisioned interface without requiring the application to re-provision the interface. However, applications must remove any HDLC channels associated with an E1 link before invoking **NMS_V5DeleteE1** for that link.

When the application invokes **NMS_V5DeleteE1**, the NMS V5 library stops monitoring status information associated with the physical E1 link. Applications cannot call **NMS_V5DeleteE1** on standby variants.

Applications provide the following information in the NMS_V5_E1_LOCATION_T structure:

Field	Description
boardNb	Logical board number where the E1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the E1 link (as defined by NMS OAM).

See also

NMS_V5ProvisionInterface, **NMS_V5AddE1**

Example

```
void  DeleteE1( void )
{
    NMS_V5_RESULT_T       NmsResult;
    DWORD                 InterfaceId;
    NMS_V5_E1_LOCATION_T  NMSE1Loc;

    printf("NMS_V5DeleteE1:\n");

    PrintInterfaces();
    /* Get parameters */
    promptdw_nodft("Enter interfaceId", &InterfaceId);

    /* Ask for E1 location parameters */
    GetE1LocationNMS( &NMSE1Loc );

    NmsResult = NMS_V5DeleteE1( InterfaceId, NMSE1Loc );

    printf ("NMS_V5DeleteE1: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5DestroyInterface

Destroys both active and standby variants of a specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5DestroyInterface** (NMS_V5_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5DestroyInterface internally destroys all on-board connections for the HDLC channels and E1 links associated with the active variant, and clears any internal structures allocated for the specified interface for both active and standby variants.

Applications must stop the interface (with **NMS_V5StopInterface**) before invoking **NMS_V5DestroyInterface**, and must invoke **NMS_V5DestroyInterface** before re-provisioning the interface with **NMS_V5ProvisionInterface**.

See also

NMS_V5DestroyStandByVariant

Example

```
void  DestroyInterface( void )
{
    NMS_V5_RESULT_TNmsResult;
    DWORD          InterfaceId;

    printf("NMS_V5DestroyInterface:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    NmsResult = NMS_V5DestroyInterface( InterfaceId );

    printf ("NMS_V5DestroyInterface:
      Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5DestroyStandByVariant

Destroys a provisioned standby variant on the specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5DestroyStandByVariant** (
NMS_V5_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5DestroyStandByVariant destroys the specified interface and clears the internal structures allocated for the standby variant on the specified provisioned interface.

See also

NMS_V5DestroyInterface, **NMS_V5ProvisionStandByVariant**

Example

```
void  DestroyStandbyVariant( void )
{
       NMS_V5_RESULT_TNmsResult;
       DWORD          InterfaceId;

       printf("NMS_V5DestroyStandbyVariant:\n");

       PrintInterfaces();
       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       NmsResult = NMS_V5DestroyStandbyVariant( InterfaceId );

       printf ("NMS_V5DestroyStandbyVariant:
              Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5Exit

Performs an internal V5 library clean-up procedure prior to exiting the library.

Prototype

NMS_V5_RESULT_T **NMS_V5Exit**

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5Exit stops and destroys all HDLC channels and E1 links provisioned by the application, flushes trace buffers, stops NMS V5 library tasks, and frees internally allocated memory.

See also

NMS_V5DestroyInterface, **NMS_V5StopInterface**

Example

```
void  Exit( void )
{
    NMS_V5_RESULT_T NmsResult;

    printf("NMS_V5Exit:\n");

    NmsResult = NMS_V5Exit();
    printf ("NMS_V5Exit: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5GetChannelStatistics

Returns the statistics collected for a particular HDLC channel provisioned on an interface.

Prototype

NMS_V5_RESULT_T **NMS_V5GetChannelStatistics** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_CHANNEL_LOCATION_T ***channel_loc,*** NMS_V5_CHANNEL_STATISTICS_T ****channel_stat***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location for which to obtain statistics as specified in the following structure:
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD boardNb;`
	`  DWORD trunkNb;`
	`  DWORD timeslotNb;`
	`  } CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

channel_stat	Pointer to an application-provided buffer to receive the following structure:
	`typedef struct {`
	`  DWORD Size;`
	`  CHANNEL_STATISTICS_RECEIVE_T chan_stat_rx;`
	`  CHANNEL_STATISTICS_TRANSMIT_T chan_stat_tx;`
	`} NMS_V5_CHANNEL_STATISTICS_T;`
	
	`typedef struct {`
	`  DWORD Octets;`
	`  DWORD Frames;`
	`  DWORD Drops;`
	`  DWORD FifoOverruns;`
	`  DWORD Aborts;`
	`  DWORD CrcErrors;`
	`  DWORD NonAlignedOctets;`
	`  DWORD BufferOverflows;`
	`} CHANNEL_STATISTICS_RECEIVE_T;`
	
	`typedef struct {`
	`  DWORD Octets;`
	`  DWORD Frames;`
	`  DWORD Drops;`
	`  DWORD FifoUnderruns;`
	`  DWORD FifoOverruns;`
	`} CHANNEL_STATISTICS_TRANSMIT_T;`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_CHANNEL	Specified HDLC channel is not provisioned on the specified interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_PARMS	***channel_stat*** buffer is NULL.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5GetChannelStatistics returns statistics for the provisioned HDLC channel. Applications can only obtain statistics for HDLC channels that are configured on the active variant of the specified interface.

CHANNEL_STATISTICS_RECEIVE_T returns the following information:

Field	Description
Octets	Bytes received count.
Frames	Frames received count.
Drops	Frames dropped count. Not supported on CG 6100C or CG 6500C boards.
FifoOverruns	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.
Aborts	Receive abort count.
CrcErrors	Receive CRC errors count.
NonAlignedOctets	Non-aligned octets count. Not supported on CG 6100C or CG 6500C boards.
BufferOverflows	Buffer overflows count. Not supported on CG 6100C or CG 6500C boards.

CHANNEL_STATISTICS_TRANSMIT_T returns the following information:

Field	Description
Octets	Bytes transmitted count.
Frames	Frames transmitted count.
Drops	Frames dropped count.
FifoUnderruns	FIFO underrun count. Not supported on CG 6100C or CG 6500C boards.
FifoOverruns	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.

See also

NMS_V5GetE1Status, **NMS_V5ResetChannelStatistics**

Example

```
void  GetChannelStatistics( void )
{
    NMS_V5_RESULT_T             NmsResult;
    DWORD                       InterfaceId;
    NMS_V5_CHANNEL_STATISTICS_T  Statistics;
    NMS_V5_CHANNEL_LOCATION_T    ChannelLocation;

    printf("NMS_V5GetChannelStatistics:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    printf("Enter channel location:\n");
    GetChannelLocationNMS( &ChannelLocation );

    NmsResult = NMS_V5GetChannelStatistics ( InterfaceId,
                                             ChannelLocation,
                                             &Statistics);

    printf ("NMS_V5GetChannelStatistics:
             Result=%s\n",PRINT_RESULT(NmsResult));
}

  if (NmsResult == NMSV5_SUCCESS)
  {
    printf("TxOctets        =%i\n", Statistics.chan_stat_tx.Octets);
    printf("TxFrames        =%i\n", Statistics.chan_stat_tx.Frames);
    printf("TxDrops         =%i\n", Statistics.chan_stat_tx.Drops);
    printf("TxUnderrun      =%i\n", Statistics.chan_stat_tx.FifoUnderruns);
    printf("TxOverrun       =%i\n", Statistics.chan_stat_tx.FifoOverruns);
    printf("RxOctets        =%i\n", Statistics.chan_stat_rx.Octets);
    printf("RxFrames        =%i\n", Statistics.chan_stat_rx.Frames);
    printf("RxDrops         =%i\n", Statistics.chan_stat_rx.Drops);
    printf("RxOverrun       =%i\n", Statistics.chan_stat_rx.FifoOverruns);
    printf("RxAbort         =%i\n", Statistics.chan_stat_rx.Aborts);
    printf("RxCrcError      =%i\n", Statistics.chan_stat_rx.CrcErrors);
    printf("RxNonAligned    =%i\n", Statistics.chan_stat_rx.NonAlignedOctets);
    printf("RxBufferOverflow=%i\n", Statistics.chan_stat_rx.BufferOverflows);
    }
}
```

NMS_V5GetE1Status

Returns the status information for a specified E1 link provisioned on an active variant of an interface.

Prototype

NMS_V5_RESULT_T **NMS_V5GetE1Status** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_E1_LOCATION_T ***e1_loc***, NMS_V5_E1_STATUS_T ****e1_status***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

e1_loc	E1 link location specified in the following structure:
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`} CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

e1_status	Pointer to an application-provided buffer for receiving the following structure:
	`typedef struct {`
	`  DWORD Size;`
	`  NMS_V5_E1_STATUSMASK_T StatusMask;`
	`  DWORD Slips;`
	`  DWORD Es;`
	`  DWORD Ses;`
	`  DWORD Uas;`
	`  DWORD LineErrors;`
	`  DWORD FrameErrors;`
	`  DWORD ElaspedTime;`
	`} NMS_V5_E1_STATUS_T;`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_E1	E1 link is not provisioned on the interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_PARMS	***channel_stat*** buffer is NULL.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5GetE1Status gets status information about a specified E1 link. The E1 link must be provisioned on an active variant of the interface. All status information values represent counts of events that occur within a time interval defined by the Elasped_time parameter.

The NMS_V5_E1_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where the E1 link is located (as defined by NMS OAM).
trunkNb	Trunk number associated with the E1 link as defined by NMS OAM.

The NMS_V5_E1_STATUS_T structure returns the following data:

Field	Description
Size	Size of the structure.
StatusMask	Current E1 status bit mask. This can be a combination of the following values: NMSV5_E1_LOS Loss of signal. NMSV5_E1_LOF Loss of frame. NMSV5_E1_AIS Alarm indication signal. NMSV5_E1_RAI Remote alarm indicator. NMSV5_E1_CRC_BLOCK_ERR CRC block error. NMSV5_E1_CRC_BLOCK_INFO CRC block information (FEBE). NMSV5_E1_NORMAL_FRAMES_0 Normal E1 frames (remote SA7 = 0). NMSV5_E1_NORMAL_FRAMES_1 Normal E1 frames (remote SA7 = 1).
Slips	Slip count.
Es	Errored seconds count.
Ses	Severely errored seconds count.
Uas	Unavailable seconds count.
LineErrors	Line code violation count.
FrameErrors	Frame bit error and CRC error count.
ElaspedTime	Seconds since counters started. This represents the duration of the observation period (in seconds).

See also

NMS_V5GetChannelStatistics, **NMS_V5ResetE1Status**

Example

```
void  GetE1Status( void )
{
    NMS_V5_RESULT_T         NmsResult;
    DWORD                   InterfaceId;
    NMS_V5_E1_LOCATION_T    E1Location;
    NMS_V5_E1_STATUS_T      E1Status = {0};

    printf("NMS_V5GetE1Status:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);
    printf("Enter E1 location:\n");
    GetE1LocationNMS( &E1Location );

    NmsResult = NMS_V5GetE1Status( InterfaceId,
                                   E1Location,
                                   &E1Status);
    printf ("NMS_V5GetE1Status:Result=%s\n", PRINT_RESULT(NmsResult));

    if (NmsResult == NMSV5_SUCCESS)
    {
        printf("StatusMask: LOS=%i LOF=%i AIS=%i RAI=%i CRCErr=%i
               FEBE=%i N_SA7_0=%i N_SA7_1=%i\n",
           ((E1Status.StatusMask & NMSV5_E1_LOS) ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_LOF) ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_AIS) ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_RAI) ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_CRC_BLOCK_ERR)   ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_CRC_BLOCK_INFO)  ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_NORMAL_FRAMES_0) ? 1 : 0),
           ((E1Status.StatusMask & NMSV5_E1_NORMAL_FRAMES_1) ? 1 : 0));

        printf("Slips           =%i\n",   E1Status.Slips);
        printf("Es              =%i\n",   E1Status.Es);
        printf("Ses             =%i\n",   E1Status.Ses);
        printf("Uas             =%i\n",   E1Status.Uas);
        printf("LineErrors      =%i\n",   E1Status.LineErrors);
        printf("FrameErrors     =%i\n",   E1Status.FrameErrors);
        printf("ElaspedTime     =%i\n",   E1Status.ElaspedTime);
    }
}
```

NMS_V5Initialize

Loads and initializes the NMS V5 library.

Prototype

NMS_V5_RESULT_T **NMS_V5Initialize** (NMS_V5_BOARD_FAMILY_T ***board_family***, void ****init_parms***)

Argument	Description

board_family	Specifies the board family used on the system. The NMS_V5_BOARD_FAMILY_T type is defined in the following structure:
	`typedef enum _NMS_V5_BOARD_FAMILY_T`
	`{`
	`  NMSV5_BOARD_FAMILY_CG = 0x01`
	`} NMS_V5_BOARD_FAMILY_T;`

init_parms	Pointer to board-specific initialization parameters. For CG 6000 and CG 6000C boards, the application passes NULL. For CG 6100C and CG 6500C boards, the application passes a CTA context handle on which the Switching (SWI) service has been opened.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_ALREADY_INITIALIZED	NMS V5 library is already initialized.
NMSV5_FAILED_LOAD_LIB	Failed to load board specific implementation library.
NMSV5_INTERNAL_FAILURE	Failed to initialize internal tasks.
NMSV5_INVALID_PARMS	Invalid function arguments.
NMSV5_NO_MEMORY	Failed to allocate memory.

Details

NMS_V5Initialize loads and initializes the NMS V5 library. Applications invoke this function only once at start-up time, and before invoking any other NMS V5 library functions.

If the system uses CG 6000 or CG 6000C boards, the application passes NULL as the ***init_parms*** argument.

When invoking **NMS_V5Initialize** on systems that use CG 6100C or CG 6500C boards, applications use the ***init_parms*** argument to pass a pointer to a CTA context handle on which the Switching service has been opened. When CG 6100C or CG 6500C boards reside on the system, applications must perform the following Natural Access operations before initializing the NMS V5 library:

- Invoke **ctaInitialize** with the list of services to initialize, including the Switching service.
- Invoke **ctaCreateQueue** to create an event queue.
- Invoke **ctaCreateContext** to create a context.
- Invoke **ctaOpenServices to** open appropriate Natural Access services, including the Switching service, on the context.

  ctaOpenServices returns a context handle that the application passes to the NMS V5 library when invoking **NMS_V5Initialize**.

If the application wants to receive tracing information from the NMS V5 library, it must invoke **NMS_V5SetTrace** immediately after invoking **NMS_V5Initialize**.

See also

NMS_V5Exit, **NMS_V5ProvisionInterface**

Example

```
void  Initialize( void* InitParms )
{
    NMS_V5_RESULT_T  NmsResult;

    printf("NMS_V5Initialize:\n");

    NmsResult = NMS_V5Initialize( g_BoardFamily, InitParms );
    printf ("NMS_V5Initialize: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5PhSendData

Sends a LAPV5 frame to a specified HDLC channel.

Prototype

NMS_V5_RESULT_T **NMS_V5PhSendData** (NMS_GR303_INTERFACE_ID_T ***interfaceId***, NMS_V5_CHANNEL_LOCATION_T ***channel_loc***, DWORD ***number_bytes***, void ***lapv_data***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	HDLC channel location to which to send data.
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`}  CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`

number_bytes	Size of LAPV frame in bytes. The maximum number of bytes is 260.
lapv_data	Pointer to an application buffer containing the LAPV frame to send.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_CHANNEL	Specified HDLC channel is not provisioned on this channel.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_PARMS	Invalid function arguments.
NMSV5_INVALID_STATE	Specified interface is not started.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Events

Event	Description
NMSV5_EVENT_TX_ERROR	Internal HDLC transmit error occurred (no buffer is returned). Returned by a **NMS_V5PhSendData** call.
NMSV5_EVENT_TX_FIFO_UNDERRUN	HDLC transmit FIFO underrun. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_FIFO_OVERRUN	HDLC transmit FIFO overrun. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_FIFO_QUEUE_FULL	HDLC transmit send queue is full (no buffer is returned). This error should not occur under normal circumstances.

Details

NMS_V5PhSendData is asynchronous and returns without waiting for the board to report the result of the send operation. The application must start the interface before it calls **NMS_V5PhSendData**.

If an error occurs during the transmission, the error is reported to the application by the following application-defined callback:

```
typedef void( *NMS_V5_CHANNEL_CALLBACK_T )
   ( NMS_V5_INTERFACE_ID_T interfaceId,
     NMS_V5_CHANNEL_LOCATION_T channel_loc,
     NMS_V5_CHANNEL_EVENT_T channel_event,
     void *databuffer,
     DWORD datasize);
```

To return error information, the NMS V5 library invokes an application-defined callback (**channel_callback**) and specifies the type of transmit error in the channel_event field. When the NMS V5 library returns a transmission error, it does not indicate the send request with which the error is associated.

Note: Results of send operations are not reported on CG 6100C boards.

See also

NMS_V5StartInterface

Example

```
void SendData(BYTE  *FrameBuffer, DWORD NumberBytes )
{
NMS_V5_RESULT_T           NmsResult;
DWORD                     InterfaceId;
NMS_V5_CHANNEL_LOCATION_T ChannelLocation;

printf("SendData:\n");

/* Get parameters */
promptdw_nodft("Enter interfaceId", &InterfaceId);

printf("Enter channel location:\n");
GetChannelLocationNMS( &ChannelLocation );

NmsResult = NMS_V5PhSendData( InterfaceId,
                              ChannelLocation,
                              NumberBytes,
                              FrameBuffer );
printf ("PhSendData: Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5ProvisionInterface

Configures the HDLC channels and E1 links for an active variant on the specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5ProvisionInterface** (NMS_V5_INTERFACE_ID_T ***interfaceId*** DWORD ***num_e1s*** NMS_V5_E1_LOCATION_T ****e1_loc_array*** NMS_V5_E1_STATUS_CALLBACK_T ****e1_status_callback*** DWORD ***numchannels*** NMS_V5_CHANNEL_LOCATION_T ****channel_loc_array*** NMS_V5_CHANNEL_CALLBACK_T ****channel_callback*** void ****channel_rx_buffer*** DWORD ***channel_rx_buffersize***)

Argument	Description
interfaceId	Interface ID. The valid range for interface IDs is 0 - 253.
num_e1s	Number of E1 links to provision (the maximum number is 16).

e1_loc_array	Pointer to an array of NMS_V5_E1_LOCATION_T structures where each structure defines the location of an E1 link. The size of the array is specified by the ***num_e1s*** value.
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  } CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

e1_status_callback	Pointer to an application-defined E1 status callback function. The NMS V5 library calls this function every time it needs to pass an event for a provisioned E1 link to the application.
	The callback function is defined as follows:
	`typedef void( *NMS_V5_E1_STATUS_CALLBACK_T )`
	`(NMS_V5_INTERFACE_ID_T interfaceId,`
	`NMS_V5_E1_LOCATION_T channel_loc,`
	`NMS_V5_E1_STATUSMASK_T e1_status) ;`
	See the Details section for field descriptions.

numchannels	Number of HDLC channels provisioned on the specified interface (the maximum is 48).

channel_loc_array	Pointer to an array of NMS_V5_CHANNEL_LOCATION_T structures. The size of the array is specified by ***num_channels***. Each element of ***channel_loc_array*** defines the HDLC location:
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`  } CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

channel_callback	Pointer to an application-defined callback function. The library invokes the callback every time it needs to pass an event for a provisioned HDLC channel. The application must then return from the callback as soon as

Argument	Description

	possible. The callback function is defined as follows:
	`typedef void( *NMS_V5_CHANNEL_CALLBACK_T )`
	`( NMS_V5_INTERFACE_ID_T interfaceId,`
	`  NMS_V5_CHANNEL_LOCATION_T channel_loc,`
	`  NMS_V5_CHANNEL_EVENT_T channel_event,`
	`  void *databuffer,`
	`  DWORD datasize);`
	See the Details section for field descriptions.

channel_rx_buffer	Pointer to application-allocated memory for passing HDLC channel event data.
channel_rx_buffersize	Size of the allocated buffer specified by ***channel_rx_buffer***. The minimum requirement for the ***channel_rx_buffersize*** is defined by NMS_V5_RX_BUFFER_SIZE.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_ALREADY_INITIALIZED	Specified interface is already initialized.
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_PARMS	One or more of the function arguments are invalid.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5ProvisionInterface provisions a new active interface. **NMS_V5ProvisionInterface** creates, initializes, and resets the specified HDLC channels and E1 links on the specified boards, leaving the channels configured but disabled.

To re-provision the configuration, the application must call **NMS_V5DestroyInterface** to destroy the existing configuration before calling **NMS_V5ProvisionInterface**. However, an application can add or delete E1 links associated with voice channels without re-provisioning the entire interface by invoking **NMS_V5AddE1** and **NMS_V5DeleteE1**.

The ***e1_loc_array*** argument can include not only E1 links associated with HDLC channels, but also E1 links associated with voice channels. The NMS V5 library returns E1 events (through an application-defined ***e1_status*** callback) for all links in the list.

Each NMS_V5_E1_LOCATION_T structure includes the following values:

Field	Description
boardNb	Logical board number where an E1 link is located (as defined by NMS OAM).
trunkNb	Physical trunk number associated with the E1 link (as defined by NMS OAM).

When defining an E1 status callback, the application must specify the following arguments:

Argument	Description
interfaceId	Interface ID where the E1 link is provisioned.
e1_loc	E1 link location structure.
status_event	E1 link event bit mask.

The status_event field returned by the ***e1_status_callback*** function can combine any of the following values (which applications can extract with the & operator):

Value	Description
NMSV5_E1_LOS	Loss of signal.
NMSV5_E1_LOF	Loss of frame.
NMSV5_E1_AIS	Alarm indication signal.
NMSV5_E1_RAI	Remote alarm indication signal.
NMSV5_E1_CRC_BLOCK_ERR	CRC block error.
NMSV5_E1_CRC_BLOCK_INFO	Remote CRC block info (FEBE).
NMSV5_E1_NORMAL_FRAMES_0	Normal E1 frames, remote SA7 = 0.
NMSV5_E1_NORMAL_FRAMES_1	Normal E1 frames, remote SA7 = 1.

Note: The application should return from the ***e1_status*** callback as soon as possible.

The NMS_V5_CHANNEL_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC channel (as defined by NMS OAM).
timeslotNb	Physical timeslot associated with the HDLC channel.

When defining the ***channel_callback*** function, the application must specify the following arguments:

Argument	Description
interfaceId	Interface ID upon which the HDLC channel is provisioned.
channel_loc	HDLC location structure.
channel_event	HDLC channel event value, as defined in the table that follows.
databuffer	A pointer to the buffer attached to the event. When the channel callback function returns, the library assumes that the application finished processing the transferred data buffer, and it can overwrite the buffer.
datasize	Size of the data passed in the buffer (0 if no data).

The channel_event field in the ***channel_callback*** function can include the following values:

Value	Description
NMSV5_EVENT_RX_ABORT	The HDLC channel received an ABORT sequence. No data is available and no buffer is returned.
NMSV5_EVENT_RX_BUFFER_OVERFLOW	An internal HDLC receive buffer overflow occurred. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_CRC_ERROR	HDLC detected a bad CRC for a frame. No data is available and no buffer is returned.
NMSV5_EVENT_RX_ERROR	An internal HDLC receive error occurred. No data is available and no buffer is returned.
NMSV5_EVENT_RX_FIFO_OVERRUN	The HDLC channel received a FIFO overrun. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_MAX_LENGTH_ VIOLATION	Not supported.
NMSV5_EVENT_RX_NON_ALIGNED_OCTET	The HDLC received an incomplete frame. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_SUCCESS	A new HDLC frame was received and is available inside ***databuffer***. ***datasize*** is set to the frame size.
NMSV5_EVENT_TX_ERROR	An internal HDLC transmit error occurred. No buffer is returned.
NMSV5_EVENT_TX_FIFO_OVERRUN	An HDLC channel FIFO overrun occurred. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_FIFO_UNDERRUN	An HDLC transmit FIFO underrun occurred. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_QUEUE_FULL	The HDLC transmit send queue is full. Under normal conditions this should never occur. No buffer is returned.

See also

NMS_V5ProvisionStandByVariant, **NMS_V5StartInterface**

Example

```
void  ProvisionInterface( void )
{
    NMS_V5_INTERFACE   NewInterface = {0};
    NMS_V5_RESULT_T    NmsResult;
    BYTE               szMessage[NMS_V5_MIN_TRACE_BUF_SIZE] = {0};
    char               Selection;

    printf("NMS_V5ProvisionInterface:\n");

    /* Set parameters */
    promptdw_nodft("Enter InterfaceId", &NewInterface.InterfaceId);

    NewInterface.ChannelRxBufferSize = NMS_V5_RX_BUFFER_SIZE;
    NewInterface.ChannelRxBuffer = calloc(
                                NewInterface.ChannelRxBufferSize, 1 );

    /* Configure E1 location parameters */
     do
       {
       Selection = 'y';
       promptchar("Add a new E1? (y/n)", &Selection );
          if(Selection == 'y')
          {
             GetE1LocationNMS(
               &NewInterface.NMS_E1s[NewInterface.NumE1s++] );
          }
       }
       while (Selection != 'n' && NewInterface.NumE1s <
               NMS_V5_E1_LOCATIONS);

    /* Add channel parameters */
    do
    {
        Selection = 'y';
        promptchar("Add a new channel ? (y/n)", &Selection );
        if(Selection == 'y')
        {

         GetChannelLocationNMS(
            &NewInterface.NMS_Channels[NewInterface.NumChannels++] );
        }
}
while (Selection != 'n' && NewInterface.NumChannels <
       NMS_V5_CHANNEL_LOCATIONS);

NmsResult = NMS_V5ProvisionInterface (NewInterface.InterfaceId,
                                NewInterface.NumE1s,
                                NewInterface.NMS_E1s,
                                E1StatusCallBackFunction,
                                NewInterface.NumChannels,
                                NewInterface.NMS_Channels,
                                ChannelCallBackFunction,
                                NewInterface.ChannelRxBuffer,
                                NewInterface.ChannelRxBufferSize);

printf ("NMS_V5ProvisionInterface:
       Result=%s\n",PRINT_RESULT(NmsResult));
```

NMS_V5ProvisionStandByVariant

Configures (provisions) the HDLC channels and E1 links for a standby variant on the specified interface.

Prototype

NMS_V5_RESULT_T **NMS_V5ProvisionStandByVariant** (NMS_V5_INTERFACE_ID_T ***interfaceId*** DWORD ***num_e1s*** NMS_V5_E1_LOCATION_T ****e1_loc_array*** NMS_V5_E1_STATUS_CALLBACK_T ***e1_status_callback*** DWORD ***numchannels*** NMS_V5_CHANNEL_LOCATION_T ****channel_loc_array*** NMS_V5_CHANNEL_CALLBACK_T ****channel_callback*** void ***channel_rx_buffer*** DWORD ***channel_rx_buffersize***)

Argument	Description
interfaceId	Interface ID. The valid range for interface IDs is 0 - 253.
num_e1s	Number of E1 links provisioned. The maximum number is 16.

e1_loc_array	Pointer to an array of NMS_V5_CHANNEL_LOCATION_T structures that each define an E1 link location. The size of the array is specified by ***num_e1s***.
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`struct {`
	`DWORD boardNb;`
	`DWORD trunkNb;`
	`} CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

e1_status_callback	Pointer to an application defined E1 status callback function. After the application switches the standby variant to active, the library calls the callback function every time it needs to pass an event to the application for a provisioned E1 link. The application should return from this function as soon as possible.
	`typedef void( *NMS_V5_E1_STATUS_CALLBACK_T )`
	`(NMS_V5_INTERFACE_ID_T interfaceId,`
	`NMS_V5_E1_LOCATION_T e1_loc,`
	`NMS_V5_E1_STATUSMASK_T e1_status) ;`
	See the Details section for field descriptions.

numchannels	Number of HDLC channels provisioned on an interface. The maximum is 48.

channel_loc_array	Pointer to an array of NMS_V5_CHANNEL_LOCATION_T structures. The size of the array should be specified by ***num_channels***. Each element of the ***channel_loc_array*** defines an HDLC channel location:
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`struct {`
	`DWORD boardNb;`
	`DWORD trunkNb;`
	`DWORD timeslotNb;`
	`} CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

channel_callback	Pointer to an application-provided channel callback function. After the standby variant is switched to active with **NMS_V5SwitchOverVariantData**, the library must call this callback function whenever it needs to pass an event for a provisioned HDLC

Argument	Description

	channel. The application should return from this function as soon as possible. The callback function is defined as follows:
	`typedef void( *NMS_V5_CHANNEL_CALLBACK_T )`
	`( NMS_V5_INTERFACE_ID_T interfaceId,`
	`  NMS_V5_CHANNEL_LOCATION_T channel_loc,`
	`  NMS_V5_CHANNEL_EVENT_T channel_event,`
	`  void *databuffer,`
	`  DWORD datasize);`
	See the Details section for field descriptions.

channel_rx_buffer	Pointer to the application allocated memory to be used by the library to pass HDLC channel event data to the application.
channel_rx_buffersize	Size of the application-allocated buffer specified by ***channel_rx_buffer***. The minimum requirement for the ***channel_rx_buffersize*** is defined by NMS_V5_RX_BUFFER_SIZE.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INVALID_INTERFACE_ID	Specified ***interfaceId*** is not provisioned.
NMSV5_INVALID_PARMS	One or more of the function arguments were invalid.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5ProvisionStandByVariant provisions a standby variant for an existing interface. If a standby variant is already defined for an interface, the initial standby interface is overwritten.

NMS_V5ProvisionStandByVariant fills in the internal standby configuration structure for a specified interface. The NMS V5 library does not create HDLC channels and E1 links for the standby variant or validate HDLC channel and E1 link locations at provision time. If the standby configuration is invalid, a failure occurs only when the application invokes **NMS_V5SwitchOverVariantData**.

It is valid for the same HDLC channels and E1 links to be present in the active and standby variants of the same interface. To change the configuration of the standby variant, the application must call **NMS_V5ProvisionStandByVariant** again.

Applications must call **NMS_V5ProvisionStandByVariant** before calling **NMS_V5SwitchOverVariantData**.

The application can include within the ***e1_loc_array*** not only the E1 links carrying the HDLC channels, but also E1 links that have voice channels only. E1 events are reported on all links in the list.

The NMS_V5_E1_LOCATION_T structure includes the following values:

Field	Description
boardNb	Logical board number where the E1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the E1 link (as defined by NMS OAM).

When defining an E1 status callback function, the application must specify the following arguments:

Argument	Description
interfaceId	Interface ID associated with the E1 link.
e1_loc	E1 link location structure.
status_event	E1 link event bit mask, which can combine status_event event masks.

The status_event field returned by ***e1_status_callback*** can combine any of the following values (which applications can extract with the & operator):

Value	Description
NMSV5_E1_LOS	Loss of signal.
NMSV5_E1_LOF	Loss of frame.
NMSV5_E1_AIS	Alarm indication signal.
NMSV5_E1_RAI	Remote alarm indication signal.
NMSV5_E1_CRC_BLOCK_ERR	CRC error.
NMSV5_E1_CRC_BLOCK_INFO	Remote CRC error (FEBE).
NMSV5_E1_NORMAL_FRAMES_0	Normal E1 frames (remote SA7 = 0).
NMSV5_E1_NORMAL_FRAMES_1	Normal E1 frames (remote SA7 = 1).

The NMS_V5_CHANNEL_LOCATION_T structure includes the following values:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC channel (as defined by NMS OAM).
timeslotNb	Physical timeslot associated with the HDLC channel.

When defining a ***channel_callback*** function, the application must specify the following arguments:

Argument	Description
interfaceId	Interface ID associated with the HDLC channel.
channel_loc	HDLC location structure.
channel_event	HDLC channel event value, as defined in the table that follows.
databuffer	A pointer to the buffer attached to the event. When the channel callback function returns, the library assumes that the application finished processing the ***channel_rx_buffer***, and the application can overwrite the buffer.
datasize	Size of the data saved in the buffer (0 if no data).

The channel_event field of the ***channel_callback*** function can contain the following values:

Value	Description
NMSV5_EVENT_RX_ABORT	The HDLC interface received an ABORT sequence. No data is available and no buffer is returned.
NMSV5_EVENT_RX_BUFFER_OVERFLOW	An internal HDLC receive buffer overflow occurred. The HDLC received a frame of invalid length, and no data is available. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_CRC_ERROR	The HDLC interface detected a bad CRC for the received frame. No data is available and no buffer is returned.
NMSV5_EVENT_RX_ERROR	An internal HDLC receive error occurred. No data is available and no buffer is returned.
NMSV5_EVENT_RX_FIFO_OVERRUN	The HDLC interface received a FIFO overrun. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_MAX_LENGTH_ VIOLATION	Not supported.
NMSV5_EVENT_RX_NON_ALIGNED_OCTET	The HDLC interface received an incomplete frame. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_SUCCESS	A new HDLC frame was received and is available inside ***databuffer***. ***datasize*** is set to the frame size.
NMSV5_EVENT_TX_ERROR	An internal HDLC transmit error occurred. No data is available and no buffer is returned.
NMSV5_EVENT_TX_FIFO_OVERRUN	An HDLC channel FIFO overrun occurred. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_FIFO_UNDERRUN	An HDLC transmit FIFO underrun occurred. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_QUEUE_FULL	The HDLC transmit send queue is full. Under normal conditions this should never occur. No buffer is returned.

See also

NMS_V5DestroyStandByVariant, **NMS_V5ProvisionInterface**

Example

```
void  ProvisionStandbyVariant( void )
{
    NMS_V5_RESULT_T    NmsResult;
    NMS_V5_INTERFACE   NewInterface = {0};
    char               Selection;

    printf("NMS_V5ProvisionStandbyVariant:\n");

    /* Set parameters */
    promptdw_nodft("Enter InterfaceId", &NewInterface.InterfaceId);

    NewInterface.ChannelRxBufferSize = NMS_V5_RX_BUFFER_SIZE;
    NewInterface.ChannelRxBuffer = calloc(
    NewInterface.ChannelRxBufferSize, 1 );

    /* Add E1 location parameters */
    do
    {
      Selection = 'y';
      promptchar("Add a new E1? (y/n)", &Selection );
      if(Selection == 'y')
      {
       GetE1LocationNMS(&NewInterface.NMS_E1s[NewInterface.NumE1s++] );
      }
     }
     while (Selection != 'n' && NewInterface.NumE1s < NMS_V5_E1_LOCATIONS);

     /* Add channel parameters */
     do
     {
        Selection = 'y';
        promptchar("Add a new channel ? (y/n)", &Selection );
        if(Selection == 'y')
        {
          GetChannelLocationNMS(
           &NewInterface.NMS_Channels[NewInterface.NumChannels++]);
        }
     }
     while (Selection != 'n' && NewInterface.NumChannels <
                         NMS_V5_CHANNEL_LOCATIONS);

     NmsResult = NMS_V5ProvisionStandbyVariant (
                                      NewInterface.InterfaceId,
                                      NewInterface.NumE1s,
                                      NewInterface.NMS_E1s,
                                      E1StatusCallBackFunction,
                                      NewInterface.NumChannels,
                                      NewInterface.NMS_Channels,
                                      ChannelCallBackFunction,
                                      NewInterface.ChannelRxBuffer,
                                      NewInterface.ChannelRxBufferSize);

     printf ("NMS_V5ProvisionStandbyVariant:
              Result=%s\n",PRINT_RESULT(NmsResult));

     if( NMSV5_SUCCESS != NmsResult )
     {
        /* Discard bad interface configuration */
        free(NewInterface.ChannelRxBuffer);
     }
  }
```

NMS_V5ResetChannelStatistics

Resets the statistics collected for a specified HDLC channel provisioned on the active interface variant.

Prototype

NMS_V5_RESULT_T **NMS_V5ResetChannelStatistics** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_CHANNEL_LOCATION_T ****channel_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

channel_loc	Pointer to the HDLC channel location.
	`typedef union _NMS_V5_CHANNEL_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  DWORD  timeslotNb;`
	`  } CG;`
	`}NMS_V5_CHANNEL_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_CHANNEL	Specified HDLC channel is not provisioned on this interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5ResetChannelStatistics resets the statistics for the specified HDLC channel. To reset the statistics, the interface must be provisioned (with **NMS_V5ProvisionInterface**) and the HDLC channel must be configured on the active variant of the interface.

The NMS_V5_CHANNEL_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where an HDLC instance is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the HDLC channel (as defined by NMS OAM).
timeslotNb	Physical timeslot associated with the HDLC channel.

See also

NMS_V5GetChannelStatistics

Example

```
void  ResetChannelStatistics( void )
{
    NMS_V5_RESULT_T             NmsResult;
    DWORD                       InterfaceId;
    NMS_V5_CHANNEL_LOCATION_T   ChannelLocation;

    printf("NMS_V5ResetChannelStatistics:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    printf("Enter channel location:\n");
    GetChannelLocationNMS( &ChannelLocation );

    NmsResult = NMS_V5ResetChannelStatistics (InterfaceId,
                                              ChannelLocation);

    printf ("NMS_V5ResetChannelStatistics:
    Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5ResetE1Status

Resets status information for a specified E1 link.

Prototype

NMS_V5_RESULT_T **NMS_V5ResetE1Status** (NMS_V5_INTERFACE_ID_T ***interfaceId*** NMS_V5_CHANNEL_LOCATION_T ****e1_loc***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

e1_loc	Pointer to the E1 link location for which to reset status information.
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`struct {`
	`DWORD boardNb;`
	`DWORD trunkNb;`
	`} CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_E1	Specified E1 link is not provisioned on this interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5ResetE1Status resets status information for a specified E1 link. Applications can only reset status information for E1 links provisioned on the active variant of an interface. The NMS_V5_E1_LOCATION_T structure includes the following values:

Field	Description
boardNb	Logical board number where the E1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the E1 link (as defined by NMS OAM).

See also

NMS_V5GetE1Status

Example

```
void  ResetE1Status( void )
{
    NMS_V5_RESULT_T         NmsResult;
    DWORD                   InterfaceId;
    NMS_V5_E1_LOCATION_T    E1Location;

    printf("NMS_V5ResetE1Status:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    printf("Enter E1 location:\n");
    GetE1LocationNMS( &E1Location );

    NmsResult = NMS_V5ResetE1Status( InterfaceId, E1Location );

    printf ("NMS_V5ResetE1Status:
    Result=%s\n",PRINT_RESULT(NmsResult));

}
```

NMS_V5SendSA7Bit

Sets the value of the SA7 E1 framing bit to 0 or 1 for an E1 link on a provisioned interface.

Prototype

NMS_V5_RESULT_T **NMS_V5SendSA7Bit** (NMS_V5_INTERFACE_ID_T ***interfaceId***, NMS_V5_E1_LOCATION_T ****e1_loc,*** NMS_V5_SA7_VALUE_T ***sa7_value***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

e1_loc	Pointer to the E1 link location specified in the following structure:
	`typedef union _NMS_V5_E1_LOCATION_T`
	`{`
	`  struct {`
	`  DWORD  boardNb;`
	`  DWORD  trunkNb;`
	`  } CG;`
	`}NMS_V5_E1_LOCATION_T`
	See the Details section for field descriptions.

sa7_value	SA7 bit value to set (Boolean). Accepted values are 0 or 1.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_E1	Specified E1 link is not provisioned on this interface.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5SendSA7Bit is asynchronous and returns when the SA7 bit is set for the specified E1 link. Applications can only call **NMS_V5SendSA7Bit** for E1 links that are provisioned on the active interface variant.

The NMS_V5_E1_LOCATION_T structure includes the following fields:

Field	Description
boardNb	Logical board number where an E1 link is located (as defined by NMS OAM).
trunkNb	Logical trunk number associated with the E1 link (as defined by NMS OAM).

See also

NMS_V5ProvisionInterface

Example

```
void  SendSA7Bit( void )
{
    NMS_V5_RESULT_T        NmsResult;
    DWORD                  InterfaceId;
    NMS_V5_E1_LOCATION_T   E1Location;
    BOOL                   SA7Value;

    printf("NMS_V5SendSA7Bit:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    printf("Enter E1 location:\n");
    GetE1LocationNMS( &E1Location );
    promptdw_nodft("Enter SA7 Value (0 or 1))", &SA7Value);

    NmsResult = NMS_V5SendSA7Bit( InterfaceId, E1Location, SA7Value );
    printf ("NMS_V5SendSA7Bit: Result=%s\n",PRINT_RESULT(NmsResult));

}
```

NMS_V5SetTrace

Controls the V5 library tracing mechanism.

Prototype

NMS_V5_RESULT_T **NMS_V5SetTrace** (NMS_V5_TRACEMASK_T ***trace_mask*** NMS_V5_TRACE_CALLBACK_T ****user_trace_callback*** void ****user_trace_buffer*** DWORD ***user_trace_buffersize*** char ****user_file_name***)

Argument	Description
trace_mask	Trace mask that specifies the level of tracing performed by the NMS V5 library. Refer to the Details section for a list of trace masks that applications can specify.
user_trace_callback	Pointer to an application-defined callback function. Set this value to NULL if the library is saving tracing information to a file. The NMS V5 library uses this callback function each time it has a trace buffer to pass to the application. The callback function is defined in the following way: `typedef void (*NMS_V5_TRACE_CALLBACK_T) (char *tracebuffer, DWORD datasize);` See the Details section for field descriptions.
user_trace_buffer	Pointer to the application-allocated memory that the NMS V5 library uses to pass trace data to the application.
user_trace_buffersize	Size of the application-allocated buffer ***user_trace_buffer***. The minimum requirement for the ***user_trace_buffersize*** is set by the NMS_V5_MIN_TRACE_BUF_SIZE value.
user_file_name	Pointer to the name of a file if the NMS V5 library is logging trace information. This value specifies the size of the trace blocks that the NMS V5 library uses when writing trace information to the file.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Failed to set up tracing.
NMSV5_INVALID_PARMS	Invalid argument has been passed.
NMSV5_INVALID_STATE	Interface has not been started.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

Applications can use **NMS_V5SetTrace** to configure tracing so that the NMS V5 library returns trace information through a defined callback function or logs trace information into a specified file. The following table lists the trace masks that applications can specify:

Trace mask	Description
NMSV5_TRACEMASK_NONE	No tracing.
NMSV5_TRACEMASK_FUNCTIONS	Trace function calls and returns.
NMSV5_TRACEMASK_ERRORS	Trace errors.
NMSV5_TRACEMASK_RX_BUF	Log received HDLC frames.
NMSV5_TRACEMASK_TX_BUF	Log transmitted HDLC frames.
NMSV5_TRACEMASK_HDLCERR	Log HDLC transmit/receive errors.
NMSV5_TRACEMASK_E1_STATUS	Log E1 status reports.

When defining a callback, the application specifies the following information:

Field	Description
tracebuffer	A pointer to the beginning of the trace information inside the application-allocated memory.
datasize	The size of the data returned in the tracebuffer.

To initiate the callback mode, the application must pass the callback function pointer ***user_trace_callback***, allocate a buffer, and pass the buffer address (tracebuffer) and size (***user_trace_buffersize***) of the buffer as arguments for **NMS_V5SetTrace**.

When the NMS V5 library receives a ***user_trace_callback*** value (and this value is not NULL), it configures the callback tracing mode and ignores the ***user_file_name*** value. After the **user_trace** callback function returns, the library assumes that the application is done with the data inside the tracebuffer and it can overwrite its contents. Therefore, the application should release buffers associated with **user_trace** callback functions as soon as possible.

Applications can log trace information into a file by setting the ***user_trace_callback*** and ***tracebuffer*** values to NULL. Applications can also have the V5 library perform internal buffering before logging tracing information into the file, by setting the ***user_trace_buffersize*** value to the trace block size. ***user_trace_buffersize*** specifies the size of the application-allocated memory reserved for trace messages returned to the application. The size of ***user_trace_buffersize*** must either be greater than the value specified by the NMS_V5_MIN_TRACE_BUFFER_SIZE parameter, equal to NMS_V5_MIN_TRACE_BUFFER_SIZE, or set to zero.

If the application sets ***user_trace_buffersize*** to zero, the NMS V5 library sends each trace message to the application immediately. However, even when the NMS V5 library sends individual trace messages, the application must still reserve an amount of memory greater than or equal to the size of NMS_V5_MIN_TRACE_BUFFER_SIZE.

The NMS V5 library can return the following tracing messages with NMSV5_INTERNAL_FAILURE errors:

Message	Hex	Description
PHYERR_INVALID_DEVICE	0x01	Board failed to define the HDLC or framer device type.
PHYERR_DEVICE_USED	0x02	The specified HDLC framer already exists in the NMS V5 library.
PHYERR_INVALID_HANDLE	0x03	Invalid HDLC framer handle passed. The board does not have an HDLC device configured with this handle.
PHYERR_BOARD_MESSAGE_ FAILED	0x04	Failed to send a message to the board. Internal communication problem.
PHYERR_BOARD_RESPONSE_ TIMEOUT	0x05	The board failed to respond to a message within a specified time interval.
PHYERR_INTERNAL_FAILURE	0x06	Failed to initialize or exit the NMS V5 library.
PHYERR_INVALID_PARMS	0x07	Invalid input parameters.
PHYERR_NO_MEMORY	0x08	Failed to allocate memory.
PHYERR_ALREADY_INITIALIZED	0x09	The NMS V5 library has already been initialized.
PHYERR_NO_BUFFER_DEFINED	0x0A	No buffer provided for requested data.
PHYERR_OUTOFRESOURCE	0x0B	Cannot create more HDLC instances on the board.

Example

```
void  SetTrace( void )
{
    NMS_V5_RESULT_T      NmsResult;
    NMS_V5_TRACEMASK_T   TraceMask;
    BYTE                 FileName[64];
    DWORD                TraceBufferSize;
    char                 Selection;

    printf("NMS_V5SetTrace:\n");

    /* Set to print error mesages */

    prompthex("Enter TraceMask", &TraceMask);

    TraceBufferSize = NMS_V5_MIN_TRACE_BUF_SIZE;

    /* Allocate buffer size not less than required minimum */
    g_pTraceBuffer = calloc(TraceBufferSize, 1);

    /* Log trace information to a file */
        strcpy( FileName, "nms_v5.log" );
        NmsResult =  NMS_V5SetTrace(TraceMask,
                                    NULL,
                                    NULL,
                                    TraceBufferSize,
                                    FileName);

    printf ("NMS_V5SetTrace: Result=%s\n",PRINT_RESULT(NmsResult));

}
```

NMS_V5StartInterface

Starts a provisioned interface.

Prototype

NMS_V5_RESULT_T **NMS_V5StartInterface** (NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_STATE	Specified interface is already started.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5StartInterface enables a specified interface. Applications must call **NMS_V5Initialize** and **NMS_V5ProvisionInterface** before calling **NMS_V5StartInterface**.

After an application calls **NMS_V5StartInterface**, the V5 library returns receive and transmit HDLC channel statistics reports by means of **channel_callback** functions and E1 link status reports by means of **e1_status_callback** functions.

See also

NMS_V5StopInterface

Example

```
void  StartInterface( void )
{
    NMS_V5_RESULT_T  NmsResult;
    DWORD            InterfaceId;

    printf("NMS_V5StartInterface:\n");

    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    NmsResult = NMS_V5StartInterface( InterfaceId );

    printf ("NMS_V5StartInterface:
             Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5StopInterface

Stops a provisioned interface.

Prototype

NMS_V5_RESULT_T **NMS_V5StopInterface** (NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_INVALID_STATE	Specified interface is not started.
NMSV5_NOT_INITIALIZED	NMS V5 library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5StopInterface disables the HDLC channels and E1 links associated with the specified interface leaving the channels and links configured but inactive.

After stopping an interface, the application can restart the interface without re-provisioning it with **NMS_V5StartInterface**.

See also

NMS_V5DestroyInterface

Example

```
void  StopInterface( void )
{
    NMS_V5_RESULT_T   NmsResult;
    DWORD             InterfaceId;

    printf("NMS_V5StopInterface:\n");
    /* Get parameters */
    promptdw_nodft("Enter InterfaceId", &InterfaceId);

    NmsResult = NMS_V5StopInterface( InterfaceId );

    printf ("NMS_V5StopInterface:
            Result=%s\n",PRINT_RESULT(NmsResult));
}
```

NMS_V5SwitchOverVariantData

Switches the interface to a specified standby variant.

Prototype

NMS_V5_RESULT_T **NMS_V5SwitchOverVariantData** (
NMS_GR303_INTERFACE_ID_T ***interfaceId***)

Argument	Description
interfaceId	Interface ID of a provisioned interface.

Return values

Return value	Description
NMSV5_SUCCESS	
NMSV5_INTERNAL_FAILURE	Internal failure. Refer to the trace log for more information.
NMSV5_INVALID_INTERFACE_ID	Specified interface is not provisioned.
NMSV5_NO_STANDBY	No standby variant is provisioned for the interface.
NMSV5_NOT_INITIALIZED	Library was not initialized with **NMS_V5Initialize**.

Details

NMS_V5SwitchOverVariantData stops an interface (if the interface is started), destroys the active variant connections and configurations, and configures the standby variant as active. No standby variant is defined on the interface after the switchover process finishes. Applications should then call **NMS_V5StartInterface** to start the interface. The application must provision the standby variant (with **NMS_V5ProvisionStandByVariant**) before initiating a switchover.

If a failure occurs during any stage of the switchover process, both active and standby variants are destroyed and the application must destroy and re-provision the entire interface.

See also

NMS_V5ProvisionInterface, **NMS_V5DestroyStandByVariant**

Example

```
void  SwitchOverVariantData( void )
{
       NMS_V5_RESULT_T    NmsResult;
       DWORD              InterfaceId;

       printf("NMS_V5SwitchOverVariantData:\n");

       /* Get parameters */
       promptdw_nodft("Enter interfaceId", &InterfaceId);

       NmsResult = NMS_V5SwitchOverVariantData( InterfaceId );

       printf ("NMS_V5SwitchOverVariantData:
                Result=%s\n",PRINT_RESULT(NmsResult));
}
```

11 Demonstration programs

Using the demonstration programs

This section describes how to use the *nms303tool* and *nmsv5tool* demonstration programs provided with the NMS DLCP software. Each demonstration program is shipped as source code with a makefile.

Before you start the demonstration programs:

- Install required NMS software.
- Install and verify the CG board.
- Ensure that CT bus switching is appropriately configured.
- Compile the appropriate demonstration program (either *nms303tool* and *nmsv5tool*).

Refer to the CG board installation manual for more information about installing CG boards.

nms303tool and *nmsv5tool* are interactive menu-driven demonstration programs. Use *nms303tool* and *nmsv5tool* to:

- Verify installation and operation of the NMS DLCP software.
- Experiment with the NMS DLCP functions.
- Expose working examples of the functions for different board families.
- Provide an example of integration with Aztek Access303, Exchange303, and AV5 software.

You can build the *nms303tool* and *nmsv5tool* demonstration programs in two modes:

- Standalone mode for loop back-to-back tests and library functionality tests. Refer to *Running nms303tool in standalone mode* on page 157 and to Running nmsv5tool in standalone mode.
- Integrated mode for use with Access303 or Exchange303 software. Refer to *Running nms303tool in integrated mode* on page 161 and to Running nmsv5tool in integrated mode.

Building program executables

To build the executable for the *nms303tool* and *nmsv5tool*, NMS recommends using the provided makefiles. Use the *nmake* utility to build the executable under Windows or the *gmake* utility to build the executable under UNIX.

Before you can compile the *nms303tool* or *nmsv5tool* program executables, you must perform the following tasks:

Step	Action
1	Build the Access303, Exchange303, or AV5 library.
2	Add the location of the *303rdt* library (for the Access303 stack), *303idt* library (for the Exchange303 stack), or the *v5an* library (for the AV5 stack) to the LIB system environment variable. This is necessary so that the linker can create an executable.

Build the *nms303tool* and *nmsv5tool* executables under Windows by entering the following:

To build this utility...	Enter the following command...
nms303tool for Access303	nmake [AZTEKROOT_RDT=***path***]
nms303tool for Exchange303	nmake [AZTEKROOT_IDT=***path***]
nmsv5tool for AV5	nmake [AZTEKROOT=***path***]

where ***path*** provides the directory path to the root directory of the Aztek protocol stack. The demonstration program makefiles use the AZTEKROOT variable to find the Access303, Exchange303, or AV5 header files needed to compile the integration portion of the demonstration program code.

Example 1: Building nms303tool as a standalone program

To build a *nms303tool* executable for use with CG boards under Windows, compile a Windows standalone version of the *nms303tool* executable by entering:

```
nmake
```

To build an *nms303tool* executable for use with CG boards under UNIX:

Step	Action
1	Set the Lib environment variable to include the directory where the Access303 stack library (*303rdt*) resides (in this case */opt/nms/dlcp/access303/coreimage*), by entering the following: `setenv LIB /opt/nms/dlcp/access303/coreimage:$LIB%`
2	Compile a UNIX standalone version of the *nms303tool* executable by entering: `gmake`

Example 2: Building nmsv5tool as an integrated program

To build an *nmsv5tool* executable that is integrated with the AV5 stack under Windows, enter the following command:

```
nmake AZTEKROOT=d:\nms\dlcp\an5
```

In this example, the AV5 library (*v5an*) resides in the *d:\nms\dlcp\an5 directory*.

To build an *nmsv5tool* executable that is integrated with the AV5 stack and can be used with CG boards under UNIX, enter the following command:

```
gmake AZTEKROOT=/opt/nms/dlcp/an5
```

In this example, the AV5 library (*v5an*) resides in the */opt/nms/dlcp/an5* directory.

Running nms303tool in standalone mode

Before running *nms303tool*, make sure you have configured and booted a CG board, connected it to the appropriate T1 links, and compiled the *nms303tool* executable.

To run *nms303tool* in standalone mode:

Step	Action
1	Start *nms303tool* by entering the following command at the prompt: `nms303tool` *nms303tool* displays a main menu of available options that directly correspond to functions in the NMS GR303 library.
2	Enter options to execute functions from the NMS GR303 library. See *nms303tool options* on page 158. When necessary, *nms303tool* prompts you to enter associated arguments or parameters for the specified function.
3	To validate the HDLC connections, use option `13` (**NMS_GR303PhSendData**) to send and receive HDLC frames on the HDLC channels.
4	To exit *nms303tool*, enter `q`.

nms303tool options

Index	Option	Description
1	Call NMS_GR303Initialize	Initializes the NMS GR303 library.
2	Call NMS_GR303SetTrace	Sets the trace level for the NMS GR303 library.
3	Call NMS_GR303Exit	Exits the NMS GR303 library.
4	Call NMS_GR303ProvisionInterface	Provisions a GR303 physical layer interface through the NMS GR303 library.
5	Call NMS_GR303DestroyInterface	Destroys an existing GR303 physical layer interface through the NMS GR303 library.
6	Call NMS_GR303ModifyChannelLocation	Moves a provisioned HDLC channel (if specified) or adds a new HDLC channel to an interface without re-provisioning the interface.
7	Call NMS_GR303StartInterface	Starts a GR303 interface.
8	Call NMS_GR303StopInterface	Stops a GR303 interface.
9	Call NMS_GR303GetChannelStatistics	Retrieves channel statistics for a channel associated with a GR303 physical layer interface.
10	Call NMS_GR303ResetChannelStatistics	Resets channel statistics for a channel associated with a GR303 physical layer interface.
11	Call NMS_GR303GetDS1Status	Retrieves status information for a DS1 link associated with a GR303 physical layer interface.
12	Call NMS_GR303ResetDS1Status	Resets status information for a DS1 link associated with a GR303 physical layer interface.
13	Call NMS_GR303PhSendData	Sends test HDLC frames to an HDLC channel.
g		Sets application global trace mask. Enter one of the following options: 0x0: No trace mask. 0x1: Trace HDLC buffers. 0x2: Trace HDLC errors.
l		Starts a load test for a provisioned interface.
p		Prints the current provisioned configuration.
h		Displays a Help menu.
q		Exits the program.

Example: nms303tool in standalone mode

This example shows the *nms303tool* options used to perform the following tasks:

- Initialize the NMS GR303 library.
- Provision an interface (interface 0) consisting of four HDLC channels on a CG board. Trunks 0 and 1 must be connected with a crossover cable.
- Start the provisioned interface.

To perform these tasks:

1. Start *nms303tool* by entering the following at the command prompt:

```
nms303tool
```

2. Enter option `1` to initialize the library.

   nms303tool displays the following message:

```
NMS_GR303Initialize: Result=NMSGR303_SUCCESS
```

3. Enter option `4` to provision the interface.
4. *nms303tool* requests the following additional information:

Prompt	Response
Enter interfaceId:	0
Enter ChannelRxBufferSize (Min=260):	260
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	24
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	12
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	24
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	12

   nms303tool displays the following message when the interface is provisioned:

```
Result =NMSGR303_SUCCESS
```

5. Enter option `7` to start the interface.

   nms303tool prompts you to enter the interface ID of the interface to start.

6. Enter the interface ID of the provisioned interface (`0`).

   nms303tool displays the following message:

   ```
   Result=NMSGR303_SUCCESS
   ```

7. Enter option `p` to display the configuration of interface 0.

   nms303tool displays configuration information for interface 0:

   ```
   InterfaceId=0:
   NMS: boardNb=0 trunkNb=0 timeslotNb=24
   NMS: boardNb=0 trunkNb=0 timeslotNb=12
   NMS: boardNb=0 trunkNb=1 timeslotNb=24
   NMS: boardNb=0 trunkNb=1 timeslotNb=12
   ```

8. To validate that HDLC channels can send and receive HDLC frames over HDLC links, enter option `13` (**NMS_GR303PhSendData**).

9. Enter other command line options to test additional NMS GR303 library functions.

10. To exit *nms303tool*, enter `q`.

Running nms303tool in integrated mode

Before running *nms303tool*, make sure you have configured and booted a CG board, connected T1 trunks to the peer side of the GR303 interface, and compiled the *nms303tool* and *aim303* executables.

To run *nms303tool* integrated with Aztek Access303 or Exchange303:

Step	Action
1	Start *nms303tool* by entering the following command at the prompt: `nms303tool` *nms303tool* initializes the Access303 or Exchange303 library. Access303 starts internal tasks including AimTaskAr303, which makes it possible for *nms303tool* to communicate with the Aztek *aim303* utility. *aim303* is a part of the Aztek software and should be used with *nms303tool* integration tests. *nms303tool* displays a main menu of available options. These options directly correspond to functions in the NMS GR303 library. These options are the same options that appear if you run in standalone mode, except that **NMS_GR303PhSendData** is not available in integrated mode. See *nms303tool options* on page 158. **Note:** *nms303tool* calls **NMS_GR303PhSendData** internally when the Access303 or Exchange303 library calls Ar303PhDataRequest. All HDLC data received from the IDT or RDT side, as well as any HDLC transmit or receive errors, are passed to the Access303 or Exchange303 stack through Ar303PhDataIndication.

2	Start the *aim303* utility by entering the following command:
	`aim303 hostname`
	where ***hostname*** is the host name of the machine where *nms303tool* is running.
	nms303tool displays the following message:
	`AR303AimTask: Menu Socket Connected on port 4200`

3	Enter options to execute functions from the NMS GR303 library. When necessary, *nms303tool* prompts you to enter associated arguments or parameters for the specified function. Enter options at the *aim303* menu (for controlling Access303 or Exchange303 stack).
4	To exit *nms303tool*, enter `q`.

Example: nms303tool in integrated mode

The following example shows how to use *nms303tool* in integrated mode to:

- Initialize the NMS GR303 library.
- Provision and start an interface (interface 0) with four HDLC channels on a CG board.

This example creates the following parameter mapping between the Aztek Access303 and NMS GR303 libraries:

Aztek AV5 library parameters	NMS V5 library parameters
DS1=1, DS0=24	boardNb=0, trunkNb=0, timeslotNb=24
DS1=1, DS0=12	boardNb=0, trunkNb=0, timeslotNb=12
DS1=2, DS0=24	boardNb=0, trunkNb=1, timeslotNb=24
DS1=2, DS0=12	boardNb=0, trunkNb=1, timeslotNb=12

To perform these tasks:

1. Start *nms303tool* by entering the following command at the prompt:

   ```
   nms303tool
   ```

2. Start *aim303* by entering the following command at the prompt:

   ```
   aim303tool hostname
   ```

   where ***hostname*** specifies the system where the NMS GR303 library is running.

3. Enter option `1` to initialize the library.

   nms303tool displays the following message:

   ```
   NMS_GR303Initialize: Result=NMSGR303_SUCCESS
   ```

4. Enter option `4` to provision the interface.

nms303tool requests the following additional information:

Prompt	Response
Enter interfaceId:	0
Enter ChannelRxBufferSize (Min=260):	260
Add a new channel? (y/n):	y
Aztek DS1 (1-128):	1
Aztek DS0 (12 = EOC, 24 = TMC):	12
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	12
Add a new channel? (y/n):	y
Aztek DS1 (1-128):	1
Aztek DS0 (12 = EOC, 24 = TMC):	24
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	24
Add a new channel? (y/n):	y
Aztek DS1 (1-128):	2
Aztek DS0 (12 = EOC, 24 = TMC):	24
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	24
Add a new channel? (y/n):	y
Aztek DS1 (1-128):	2
Aztek DS0 (12 = EOC, 24 = TMC):	12
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	12

The following message displays when the interface is provisioned:

```
Result=NMSGR303_SUCCESS
```

5. Enter option `7` to start the interface.

   nms303tool prompts you to enter the interface ID of the interface to start.

6. Enter the interface ID of the provisioned interface (`0`).

   The following message displays:

```
Result=NMSGR303_SUCCESS
```

7. Enter option `p` to display the configuration of interface 0.

   nms303tool displays configuration information for interface 0:

```
InterfaceId=0:
NMS: boardNb=24 trunkNb=0 timeslotNb=24
NMS: boardNb=12 trunkNb=0 timeslotNb=12
NMS: boardNb=24 trunkNb=1 timeslotNb=24
NMS: boardNb=12 trunkNb=1 timeslotNb=12
```

8. Enter options from the *nms303tool* and *aim303* program menus to execute NMS GR303 and Access303 library functions.
9. Exit *nms303tool* by entering `q`.
10. Exit the *aim303* utility.

Running nmsv5tool in standalone mode

Before running *nmsv5tool*, make sure you have configured and booted a CG board, connected its E1 trunks, and compiled the *nmsv5tool* executable.

To run the *nmsv5tool* in standalone mode:

Step	Action
1	Start *nmsv5tool* by entering the following command at the prompt: `nmsv5tool` *nmsv5tool* displays a main menu of available options that directly correspond to functions in the NMS V5 library.
2	Enter options to execute functions from the NMS V5 library. See nmsv5tool options. When necessary, *nmsv5tool* prompts you to enter associated arguments or parameters for the specified board family.
3	To validate that the HDLC channels can send and receive HDLC frames, use option `13` (**NMS_V5PhSendData**).
4	Before exiting *nmsv5tool*, stop (option `8`) and destroy (option `5`) the interface.
5	To exit *nmsv5tool*, enter `q`.

nmsv4tool options

Index	Option	Description
1	`Call NMS_V5Initialize`	Initializes NMS V5 library.
2	`Call NMS_V5SetTrace`	Sets trace level for NMS V5 library.
3	`Call NMS_V5Exit`	Exits NMS V5 library.
4	`Call NMS_V5ProvisionInterface`	Provisions a V5.2 physical layer interface.
5	`Call NMS_V5ProvisionStandbyVariant`	Provisions a V5.2 physical layer interface standby variant.
6	`Call NMS_V5DestroyInterface`	Destroys an existing V5.2 physical layer interface.
7	`Call NMS_V5DestroyStandbyVariant`	Destroys a V5.2 physical layer interface standby variant.
8	`Call NMS_V5AddE1`	Adds an HDLC channel to a provisioned active variant on a specified interface.
9	`Call NMS_V5DeleteE1`	Removes an HDLC channel from a provisioned active variant on a specified interface.
10	`Call NMS_V5AddE1`	Adds an E1 link associated with voice channels to a provisioned active variant on a specified interface.
11	`Call NMS_V5DeleteE1`	Deletes an E1 link from a provisioned active variant on a specified interface.
12	`Call NMS_V5SwitchOverVariantData`	Switches interface to the standby variant.
13	`Call NMS_V5StartInterface`	Starts a V5.2 physical layer interface.

Index	Option	Description
14	Call NMS_V5StopInterface	Stops a V5.2 physical layer interface.
15	Call NMS_V5GetChannelStatistics	Retrieves channel statistics for an HDLC channel associated with a V5.2 physical layer interface.
16	Call NMS_V5ResetChannelStatistics	Resets channel statistics for an HDLC channel associated with a V5.2 physical layer interface.
17	Call NMS_V5GetE1Status	Retrieves status information for an E1 link associated with a V5.2 physical layer interface.
18	Call NMS_V5ResetE1Status	Resets status information for an E1 link associated with a V5.2 physical layer interface.
19	Call NMS_V5PhSendData	Sends test frames to the NMS V5 library.
20	Call NMS_V5SendSA7Bit	Sets the value of the SA7 E1 framing bit to 0 or 1.
g		Sets application global trace mask. Enter one of the following options: 0x0: No trace mask. 0x1: Trace HDLC buffers. 0x2: Trace HDLC errors. 0x4: Trace E1 status reports.
l		Starts a load test for a provisioned interface.
p		Prints the current provisioned configuration.
h		Displays a Help menu.
q		Exits the program.

Note: Options 19 and 20 (calling **NMS_V5PhSendData** and **NMS_V5SendSA7Bit**) are only available in standalone mode. When the NMS V5 library is integrated with the AV5 library, *nmsv5tool* calls **NMS_V5PhSendData** and **NMS_V5SendSA7Bit** internally as requested by the AV5 library **V5PhDataRequest**, **V5SetSA7ToOne**, and **V5SetSA7ToZero** functions.

Example: nmsv5tool in standalone mode

This example shows the *nmsv5tool* options used to perform the following tasks:

- Initialize the NMS V5 library.
- Provision an interface (interface 0) consisting of four HDLC channels on a CG board. Trunks 0 and 1 must be connected with a crossover cable.
- Start the provisioned interface.

To perform these tasks:

1. Start *nmsv5tool* by entering the following command at the prompt:

```
nmsv5tool
```

2. Enter option 1 to initialize the library.

   nmsv5tool displays the following:

```
NMS_V5Initialize: Result=NMSV5_SUCCESS
```

3. Enter option `4` to provision the interface.
4. *nmsv5tool* requests the following additional information:

Prompt	Response
Enter interfaceId:	0
Enter ChannelRxBufferSize (Min=260):	260
Add a new E1? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	0
Add a new E1? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	1
Add a new E1? (y/n):	n
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	16
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	15
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	16
Add a new channel? (y/n):	y
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	15
Add a new channel? (y/n):	n

The following message displays when the interface is provisioned:

```
Result=NMSV5_SUCCESS
```

5. Enter option `11` to start the interface.

   nmsv5tool prompts you to enter the interface ID of the interface to start.
6. Enter the interface ID of the provisioned interface (`0`).

7. After the interface is started, *nmsv5tool* prints the E1 link status for each provisioned E1 link to the screen and displays an NMSV5_SUCCESS event:

```
17:30:43E1Status Function InterfaceId=0 Status: LOS=0 LOF=0 AIS=0
RAI=0 CRCErr=0 FEBE=0 N_SA7_0=0 N_SA7_1=1

NMS: boardNb=0 trunkNb=0

17:30:43E1Status Function InterfaceId=0 Status: LOS=0 LOF=0 AIS=0
RAI=0 CRCErr=0 FEBE=0 N_SA7_0=0 N_SA7_1=1

NMS: boardNb=0 trunkNb=1

NMS_V5StartInterface: Result=NMSV5_SUCCESS
```

8. To validate that HDLC connections can send and receive HDLC frames over HDLC links, enter option `17`.
9. Enter other command line options to test NMS V5 library functions.
10. To exit *nmsv5tool*, enter `q`.

Running nmsv5tool in integrated mode

Before running *nmsv5tool*, you must configure and boot a CG board, connect its E1 trunks, and compile the *nmsv5tool* and *aimv5* executables.

To run *nmsv5tool* integrated with Aztek AV5:

Step	Action
1	Start *nmsv5tool*, entering the following command at the prompt: `nmsv5tool` *nmsv5tool* initializes the AV5 library. The AV5 library starts internal tasks including the AIM task, which makes it possible for the *aimv5* utility to communicate with the Aztek AV5 stack. *nmsv5tool* displays a main menu of available options. These options directly correspond to functions in the NMS V5 library. These options are the same options that appear if you run in standalone mode, except that **NMS_V5PhSendData** and **NMS_V5SendSA7Bit** do not appear in integrated mode. Refer to nmsv5tool options. **Note:** *nmsv5tool* calls **NMS_V5PhSendData** and **NMS_V5SendSA7Bit**. The AV5 stack calls V5PhDataRequest, V5SetSA7ToOne, and V5SetSA7ToZero. *nmsv5tool* passes all HDLC data received from the IDT side, as well as any HDLC transmit and receive errors, to the AV5 stack by using the AV5PhDataIndication function. *nmsv5tool* also sends E1 status reports to the AV5 library by calling the AV5 library function V5E1Status.
2	Start the *aimv5* utility by entering the following command: `aim` ***`hostname`*** where ***hostname*** is the host name of the machine where *nmsv5tool* is running. *nmsv5tool* displays the following message: `Aztek_waitconnect: AIM socket connected on port 5200, file descriptor 648`
3	Enter options to execute functions from the NMS V5 library. When necessary, *nmsv5tool* prompts you to enter associated arguments or parameters for the specified function. Enter options at the *aimv5* menu for controlling AV5 stack.
4	To exit *nmsv5tool*, enter `q`.

Example: nmsv5tool in integrated mode

This example shows the *nmsv5tool* options used to perform the following tasks while integrated with the V5 library:

- Initialize the NMS V5 library.
- Provision an interface (interface `0`) with two HDLC channels on a CG board using both the NMS V5 and AV5 libraries.
- Start the provisioned interface.

This example creates the following parameter mapping between the Aztek AV5 and NMS V5 libraries:

Aztek AV5 library parameters	NMS V5 library parameters
`E1=0 CChannel=16`	`boardNb=0, trunkNb=0, trunkNb=16`
`E1=1 CChannel=16`	`boardNb=0, trunkNb=1, trunkNb=16`

To perform these tasks:

1. Start *nmsv5tool* by entering the following command at the prompt:

```
nmsv5tool
```

2. Start *aimv5*, by entering the following command at the prompt:

```
aimv5
```

3. Enter option `1` to initialize the library at the *nmsv5tool* command line.

   nmsv5tool displays the following:

```
NMS_V5Initialize: Result=NMSV5_SUCCESS
```

4. Enter option `4` to provision the interface.

   nmsv5tool requests the following additional information:

Prompt	Response
Enter interfaceId:	0
Enter ChannelRxBufferSize (Min=260):	260
Add a new E1? (y/n):	y
Aztek E1 (0-15):	0
NMS boardNb:	0
NMS trunkNb:	0
Add a new E1? (y/n):	y
Aztek E1 (0-15):	1
NMS boardNb:	0
NMS trunkNb:	1
Add a new E1? (y/n):	n
Add a new channel? (y/n):	y
Aztek E1 (0-15):	0
Aztek C-Channel (15,16,or 31):	16
NMS boardNb:	0
NMS trunkNb:	0
NMS timeslotNb	16
Add a new channel? (y/n):	y
Aztek E1 (0-15):	1
Aztek C-Channel (15,16,or 31):	16
NMS boardNb:	0
NMS trunkNb:	1
NMS timeslotNb	16
Add a new channel? (y/n):	n

When the interface is provisioned, *nmsv5tool* displays the following message:

```
Result=NMSGRV5_SUCCESS
```

5. Enter option `11` to start the interface.

   nmsv5tool prompts you to enter the interface ID of the interface to start.

6. Enter the interface ID of the provisioned interface (`0`).

   nmsv5tool displays the following configuration information:

```
19:31:46E1Status Function InterfaceId=0 Status: LOS=0 LOF=0 AIS=0
RAI=0 CRCErr=0 FEBE=0 N_SA7_0=0 N_SA7_1=1

NMS: boardNb=0 trunkNb=0

V5PhDataIndication AztecReturn=0x30

19:31:46E1Status Function InterfaceId=0 Status: LOS=0 LOF=0 AIS=0
RAI=0 CRCErr=0 FEBE=0 N_SA7_0=0 N_SA7_1=1

NMS: boardNb=0 trunkNb=1

V5PhDataIndication AztecReturn=0x30
```

   nmsv5tool displays the following message:

```
Result=NMSV5_SUCCESS
```

7. Enter options from the *nmsv5tool* and *aimv5* program menus to execute NMS V5 and AV5 library functions.
8. Exit *nmsv5tool* by entering `q`.
9. Exit the *aimv5* utility.

12 Errors and events

NMS GR303 library errors

This topic provides an alphabetical and numerical summary of the NMS GR303 library error codes.

Alphabetical error summary

Error name	Hex	Decimal	Description
NMSGR303_ALREADY_INITIALIZED	0x02	2	NMS GR303 library already initialized.
NMSGR303_FAILED_LOAD_LIB	0x03	3	Failed to load the NMS GR303 library.
NMSGR303_INTERNAL_FAILURE	0x05	5	Operation failed. Refer to the trace log file for more information.
NMSGR303_INVALID_CHANNEL	0x08	8	Specified HDLC channel is not provisioned on this interface.
NMSGR303_INVALID_E1	0x09	9	Invalid DS1 location.
NMSGR303_INVALID_INTERFACE_ID	0x07	7	Interface ID is not valid.
NMSGR303_INVALID_PARMS	0x01	1	Invalid parameters.
NMSGR303_INVALID_STATE	0x0b	11	Invalid state for request.
NMSGR303_NO_MEMORY	0x04	4	Failed to allocate host memory.
NMSGR303_NOT_INITIALIZED	0x06	6	NMS GR303 library is not initialized.
NMSGR303_OUTOFRESOURCE	0x0A	10	No more resources are available.
NMSGR303_SUCCESS	0x00	0	Success.

Numerical error summary

Hex	Decimal	Error name
0x00	0	NMSGR303_SUCCESS
0x01	1	NMSGR303_INVALID_PARMS
0x02	2	NMSGR303_ALREADY_INITIALIZED
0x03	3	NMSGR303_FAILED_LOAD_LIB
0x04	4	NMSGR303_NO_MEMORY
0x05	5	NMSGR303_INTERNAL_FAILURE
0x06	6	NMSGR303_NOT_INITIALIZED
0x07	7	NMSGR303_INVALID_INTERFACE_ID
0x08	8	NMSGR303_INVALID_CHANNEL

Hex	Decimal	Error name
0x09	9	NMSGR303_INVALID_DS1
0x0A	10	NMSGR303_OUTOFRESOURCE
0x0B	11	NMSGR303_INVALID_STATE

NMS GR303 library channel callback events

This topic provides an alphabetical and numerical summary of the event messages that can be returned by NMS GR303 library callback functions.

Alphabetical channel callback event summary

Callback event	Hex	Decimal	Description
NMSGR303_EVENT_RX_ABORT	0x4	4	The HDLC channel received an ABORT sequence. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_BUFFER_ OVERFLOW	0x7	7	An internal HDLC receive buffer overflow occurred. The HDLC received a frame of invalid length. No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_CRC_ERROR	0x5	5	The HDLC channel received a bad CRC for the receive frame. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_ERROR	0x2	2	An HDLC receive error was returned. No data is available and no buffer is attached.
NMSGR303_EVENT_RX_FIFO_OVERRUN	0x3	3	A FIFO overrun occurred. No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_NON_ALIGNED_ OCTET	0x6	6	The HDLC channel received a non-aligned frame (not integer size of received bytes). No data is available and no buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_RX_SUCCESS	0x1	1	A new HDLC frame was received and is available inside ***databuffer***. ***datasize*** is set to the frame size.
NMSGR303_EVENT_TX_ERROR	0x8	8	An internal HDLC transmit error occurred (no buffer is attached).
NMSGR303_EVENT_TX_FIFO_OVERRUN	0x0a	10	An HDLC transmit FIFO overrun occurred. No buffer is attached. Not supported for CG 6100C or CG 6500C boards.

Callback event	Hex	Decimal	Description
NMSGR303_EVENT_TX_FIFO_UNDERRUN	0x9	9	An HDLC transmit FIFO underrun occurred. No buffer is attached. Not supported for CG 6100C or CG 6500C boards.
NMSGR303_EVENT_TX_QUEUE_FULL	0x0b	11	The HDLC transmit internal send queue is full. No buffer is attached. This indicates an abnormal send rate.

Numerical channel callback event summary

Hex	Decimal	Callback event
0x1	1	NMSGR303_EVENT_RX_SUCCESS
0x4	4	NMSGR303_EVENT_RX_ABORT
0x7	7	NMSGR303_EVENT_RX_BUFFER_OVERFLOW
0x5	5	NMSGR303_EVENT_RX_CRC_ERROR
0x2	2	NMSGR303_EVENT_RX_ERROR
0x3	3	NMSGR303_EVENT_RX_FIFO_OVERRUN
0x6	6	NMSGR303_EVENT_RX_NON_ALIGNED_OCTET
0x8	8	NMSGR303_EVENT_TX_ERROR
0x9	9	NMSGR303_EVENT_TX_FIFO_UNDERRUN
0xA	10	NMSGR303_EVENT_TX_FIFO_OVERRUN
0xB	11	NMSGR303_EVENT_TX_QUEUE_FULL

NMS V5 library errors

This topic provides an alphabetical and numerical summary of NMS V5 library error codes.

Alphabetical error summary

Error name	Hex	Decimal	Description
NMSV5_ALREADY_INITIALIZED	0x02	2	NMS V5 library is already initialized.
NMSV5_FAILED_LOAD_LIB	0x03	3	Failed to load the NMS V5 library.
NMSV5_INTERNAL_FAILURE	0x05	5	Operation failed. Refer to the trace log file for more information.
NMSV5_INVALID_CHANNEL	0x08	8	Invalid channel location.
NMSV5_INVALID_E1	0x09	9	Invalid E1 location.
NMSV5_INVALID_INTERFACE_ID	0x07	7	Interface ID is not valid.
NMSV5_INVALID_PARMS	0x01	1	Invalid parameters.
NMSV5_INVALID_STATE	0x0c	11	Invalid state for request.
NMSV5_NO_MEMORY	0x04	4	Failed to allocate host memory.
NMSV5_NO_STANDBY	0x0d	12	No standby variant is defined.
NMSV5_NOT_INITIALIZED	0x06	6	NMS V5 library is not initialized.
NMSV5_OUTOFRESOURCE	0x0a	10	No more resources are available.
NMSV5_SUCCESS	0x00	0	Success.

Numerical error summary

Hex	Decimal	Error name
0x00	0	NMSV5_SUCCESS
0x01	1	NMSV5_INVALID_PARMS
0x02	2	NMSV5_ALREADY_INITIALIZED
0x03	3	NMSV5_FAILED_LOAD_LIB
0x04	4	NMSV5_NO_MEMORY
0x05	5	NMSV5_INTERNAL_FAILURE
0x06	6	NMSV5_NOT_INITIALIZED
0x07	7	NMSV5_INVALID_INTERFACE_ID
0x08	8	NMSV5_INVALID_CHANNEL
0x09	9	NMSV5_INVALID_E1
0x0a	10	NMSV5_OUTOFRESOURCE
0x0c	11	NMSV5_INVALID_STATE

Hex	Decimal	Error name
0x0d	12	NMSV5_NO_STANDBY

NMS V5 library channel callback events

This topic provides an alphabetical and numerical summary of the event messages that can be returned by NMS V5 library callback functions.

Alphabetical channel callback event summary

Callback event	Hex	Decimal	Description
NMSV5_EVENT_RX_ABORT	0x4	4	HDLC receive ABORT sequence. No data is available and no buffer is returned.
NMSV5_EVENT_RX_BUFFER_ OVERFLOW	0x7	7	An internal HDLC receive buffer overflow occurred. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_CRC_ERROR	0x5	5	HDLC detected a bad CRC for a frame. No data is available and no buffer is returned.
NMSV5_EVENT_RX_ERROR	0x2	2	An internal HDLC receive error occurred. No data is available and no buffer is returned.
NMSV5_EVENT_RX_FIFO_OVERRUN	0x3	3	HDLC receive FIFO overrun. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_NON_ALIGNED_ OCTET	0x6	6	The HDLC channel received a non-aligned frame. No data is available and no buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_RX_NO_7E_FLAG	0x8	8	Not supported.
NMSV5_EVENT_RX_SUCCESS	0x1	1	A new HDLC frame was received and is available inside ***databuffer***. ***datasize*** is set to the frame size.
NMSV5_EVENT_TX_ERROR	0x09	9	Internal HDLC transmit error occurred. No buffer is returned.
NMSV5_EVENT_TX_FIFO_OVERRUN	0x0B	11	HDLC transmit FIFO overrun. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_FIFO_UNDERRUN	0x0A	10	HDLC transmit FIFO underrun. No buffer is returned. Not supported for CG 6100C or CG 6500C boards.
NMSV5_EVENT_TX_QUEUE_FULL	0x0C	12	HDLC transmit send queue is full. Under normal conditions this should never occur. No buffer is returned.

Numerical channel callback event summary

Hex	Decimal	Callback event
0x1	1	NMSV5_EVENT_RX_SUCCESS
0x2	2	NMSV5_EVENT_RX_ERROR
0x3	3	NMSV5_EVENT_RX_FIFO_OVERRUN
0x4	4	NMSV5_EVENT_RX_ABORT
0x5	5	NMSV5_EVENT_RX_CRC_ERROR
0x6	6	NMSV5_EVENT_RX_NON_ALIGNED_OCTET
0x7	7	NMSV5_EVENT_RX_BUFFER_OVERFLOW
0x9	8	NMSV5_EVENT_RX_NO_7E_FLAG
0xB	9	NMSV5_EVENT_TX_ERROR
0xC	10	NMSV5_EVENT_TX_FIFO_UNDERRUN
0xD	11	NMSV5_EVENT_TX_FIFO_OVERRUN
0xE	12	NMSV5_EVENT_TX_QUEUE_FULL

13 Structures

NMS GR303 library structures

When using the NMS GR303 library, applications can send or receive the following structures:

Structure	Description
NMS_GR303_BOARD_FAMILY_T	Applications send this structure to specify the board family (CG boards) used on the host system.
NMS_GR303_DS1_LOCATION_T	Applications send this structure to specify the location of a DS1 link on a CG board.
NMS_GR303_CHANNEL_LOCATION_T	Applications send this structure to specify the location of an HDLC channel on a CG board.
NMS_GR303_DS1_STATUS_T	The NMS GR303 library returns this structure to indicate the status of a particular DS1 link.
NMS_GR303_CHANNEL_STATISTICS_T	The NMS GR303 library returns this structure to provide statistics about the transmit or receive operations on HDLC channels.

NMS_GR303_BOARD_FAMILY_T

When applications initialize the NMS GR303 library, they specify the board family with NMS_GR303_BOARD_FAMILY_T:

```
typedef enum _NMS_GR303_BOARD_T
{
    NMS_BOARD_FAMILY_CG
} NMS_GR303_BOARD_FAMILY_T;
```

Dependent function: **NMS_GR303Initialize**

Field name	Type	Default	Description
NMS_BOARD_FAMILY_CG	NMS_BOARD_FAMILY_CG	N/A	Specifies the board family (CG boards) used on the system.

NMS_GR303_DS1_LOCATION_T

When applications obtain or reset DS1 link status information, they specify the DS1 link information through NMS_GR303_DS1_LOCATION_T:

```
typedef union _NMS_GR303_DS1_LOCATION_T
{
    struct {
        DWORD boardNb;
        DWORD trunkNb;
    } CG;
} NMS_GR303_DS1_LOCATION_T;
```

Dependent functions: **NMS_GR303GetDS1Status** or **NMS_GR303ResetDS1Status**

Field name	Type	Default	Units	Description
boardNb	DWORD	N/A	N/A	Logical board number of the board where the DS1 link is located (as defined by NMS OAM).
trunkNb	DWORD	N/A	N/A	Logical trunk number associated with the T1 link (as defined by NMS OAM).

NMS_GR303_CHANNEL_LOCATION_T

When applications obtain or reset HDLC channel statistics or modify a channel location, they specify the HDLC channel information through NMS_GR303_CHANNEL_LOCATION_T:

```
typedef union _NMS_GR303_CHANNEL_LOCATION_T
{
    struct {
        DWORD boardNb;
        DWORD trunkNb;
        DWORD timeslotNb;
    } CG;
} NMS_GR303_CHANNEL_LOCATION_T;
```

Dependent functions: **NMS_GR303GetChannelStatistics**, **NMS_GR303ResetChannelStatistics**, or **NMS_GR303ModifyChannelLocation**

Field name	Type	Default	Units	Description
boardNb	DWORD	N/A	N/A	Logical board number of the board where the HDLC instance is located as defined by NMS OAM.
trunkNb	DWORD	N/A	N/A	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	DWORD	N/A	N/A	Physical timeslot number associated with the HDLC instance.

NMS_GR303_DS1_STATUS_T

When the NMS GR303 library returns information about the status of a DS1 link, it provides the information in NMS_GR303_DS1_STATUS_T:

```
typedef struct {
    DWORD Size;
    NMS_GR303_DS1_STATUSMASK_T StatusMask;
    DWORD Slips;
    DWORD Es;
    DWORD Ses;
    DWORD Uas;
    DWORD LineErrors;
    DWORD FrameErrors;
    DWORD ElaspedTime;
} NMS_GR303_DS1_STATUS_T;
```

Dependent function: **NMS_GR303GetDS1Status**

Field name	Type	Units	Description
Size	DWORD	N/A	Size of the structure.
StatusMask	NMS_GR303_DS1_STATUSMASK_T	N/A	Current DS1 status bit mask. This can combine the following values: NMSGR303_DS1_LOS (0x01) Loss of signal. NMSGR303_DS1_LOF (0x02) Loss of frame. NMSGR303_DS1_AIS (0x04) Alarm indication signal. NMSGR303_DS1_RAI (0x08) Remote alarm indicator.
Slips	DWORD	count	Number of slips.
Es	DWORD	count	Number of errors seconds.
Ses	DWORD	count	Number of severely errored seconds.
Uas	DWORD	count	Number of unavailable seconds.
LineErrors	DWORD	count	Number of line code violations.
FrameErrors	DWORD	count	Number of frame bit errors and CRC errors.
ElaspedTime	DWORD	count	Seconds since counters started. This represents the duration of the observation time (in seconds). All statistics are the count of particular events that occur during the ElaspedTime interval.

NMS_GR303_CHANNEL_STATISTICS_T

When the NMS GR303 library returns HDLC channel statistics, it provides the information in NMS_GR303_CHANNEL_STATISTICS_T (which includes both a CHANNEL_STATISTICS_RECEIVE_T or CHANNEL_STATISTICS_TRANSMIT_T subtype):

```
typedef struct {
   DWORD Size; /* Size of this structure */
   CHANNEL_STATISTICS_RECEIVE_T  chan_stat_rx;
   CHANNEL_STATISTICS_TRANSMIT_T chan_stat_tx;
} NMS_GR303_CHANNEL_STATISTICS_T;

typedef struct {
  DWORD Octets;
   DWORD Frames;
   DWORD Drops;
   DWORD FifoOverruns;
   DWORD Aborts;
   DWORD CrcErrors;
  DWORD NonAlignedOctets;
   DWORD BufferOverflows;
} CHANNEL_STATISTICS_RECEIVE_T;

typedef struct {
   DWORD Octets;
   DWORD Frames;
   DWORD Drops;
   DWORD FifoUnderruns;
   DWORD FifoOverruns;
} CHANNEL_STATISTICS_TRANSMIT_T;
```

CHANNEL_STATISTICS_RECEIVE_T

Dependent function: **NMS_GR303GetChannelStatistics**

Field name	Type	Units	Description
Octets	DWORD	count	Bytes received count.
Frames	DWORD	count	Frames received count.
Drops	DWORD	count	Frames dropped count.
FifoOverruns	DWORD	count	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.
Aborts	DWORD	count	Receive abort count.
CrcErrors	DWORD	count	Receive CRC errors count.
NonAlignedOctets	DWORD	count	Non-aligned octets count. Not supported on CG 6100C or CG 6500C boards.
BufferOverflows	DWORD	count	Buffer overflows count. Not supported on CG 6100C or CG 6500C boards.

CHANNEL_STATISTICS_TRANSMIT_T

Dependent function: **NMS_GR303GetChannelStatistics**

Field name	Type	Units	Description
Octets	DWORD	count	Bytes transmitted count.
Frames	DWORD	count	Frames transmitted count.
Drops	DWORD	count	Frames dropped count.
FifoUnderruns	DWORD	count	FIFO underrun count. Not supported on CG 6100C or CG 6500C boards.
FifoOverruns	DWORD	count	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.

NMS V5 library structures

When using the NMS V5 library, applications can send or receive the following structures:

Structure	Description
NMS_V5_BOARD_FAMILY_T	Applications send this structure to specify the board family (CG boards) used on the host system.
NMS_V5_E1_LOCATION_T	Applications send this structure to specify the location of an E1 link on a CG board.
NMS_V5_CHANNEL_LOCATION_T	Applications send this structure to specify the location of an HDLC channel on a CG board.
NMS_V5_E1_STATUS_T	The NMS V5 library returns this structure to indicate the status of a particular E1 link.
NMS_V5_CHANNEL_STATISTICS_T	The NMS V5 library returns this structure to provide statistics about the transmit or receive operations on HDLC channels.

NMS_V5_BOARD_FAMILY_T

When applications initialize the NMS V5 library, they specify the board family with the NMS_V5_BOARD_FAMILY_T structure:

```
typedef enum _NMS_V5_BOARD_T
{
    NMS_BOARD_FAMILY_CG
} NMS_V5_BOARD_FAMILY_T;
```

Dependent function: **NMS_V5Initialize**

Field name	Type	Default	Description
NMS_BOARD_FAMILY_CG	NMS_V5_BOARD_FAMILY_T	N/A	Specifies the board family (CG boards) used on the system.

NMS_V5_E1_LOCATION_T

When applications obtain or reset E1 link status information, they specify the E1 link information through NMS_V5_E1_LOCATION_T:

```
typedef union _NMS_V5_E1_LOCATION_T
{
    struct {
        DWORD boardNb;
        DWORD trunkNb;
    } CG;
} NMS_V5_E1_LOCATION_T;
```

Dependent function: **NMS_V5ProvisionInterface**, **NMS_V5ProvisionStandByVariant**, **NMS_V5AddE1**, **NMS_V5DeleteE1**, **NMS_V5GetE1Status**, **NMS_V5ResetE1Status**, or **NMS_V5SendSA7Bit**

Field name	Type	Default	Units	Description
boardNb	DWORD	N/A	N/A	Logical board number of the board where the E1 link is located (as defined by NMS OAM).
trunkNb	DWORD	N/A	N/A	Logical trunk number associated with the E1 link (as defined by NMS OAM).

NMS_V5_CHANNEL_LOCATION_T

When applications obtain or reset HDLC channel statistics or modify a channel location, they specify the HDLC channel information through a NMS_V5_CHANNEL_LOCATION_T structure:

```
typedef union _NMS_V5_CHANNEL_LOCATION_T
{
    struct {
        DWORD boardNb;
        DWORD trunkNb;
        DWORD timeslotNb;
    } CG;
} NMS_V5_CHANNEL_LOCATION_T;
```

Dependent functions: **NMS_V5ProvisionInterface**, **NMS_V5ProvisionStandByVariant**, **NMS_V5PhSendData**, **NMS_V5GetChannelStatistics**, or **NMS_V5SendSA7Bit**

Field name	Type	Default	Units	Description
boardNb	DWORD	N/A	N/A	Logical board number of the board where the HDLC instance is located as defined by NMS OAM.
trunkNb	DWORD	N/A	N/A	Logical trunk number associated with the HDLC instance (as defined by NMS OAM).
timeslotNb	DWORD	N/A	N/A	Physical timeslot number associated with the HDLC instance.

NMS_V5_E1_STATUS_T

When the NMS V5 library returns information about the status of an E1 link, it provides the information in a NMS_V5_E1_STATUS_T structure:

```
typedef struct {
    DWORD Size;
    NMS_V5_E1_STATUSMASK_T StatusMask;
    DWORD Slips;
    DWORD Es;
    DWORD Ses;
    DWORD Uas;
    DWORD LineErrors;
    DWORD FrameErrors;
    DWORD ElaspedTime;
} NMS_V5_E1_STATUS_T;
```

Dependent function: **NMS_V5GetE1Status**

Field name	Type	Units	Description
Size	DWORD	N/A	Size of the structure.

Field name	Type	Units	Description
StatusMask	NMS_V5_E1_ STATUSMASK_T	N/A	Current E1 status bit mask. This can combine the following values: NMSV5_E1_LOS (0x01) Loss of signal. NMSV5_E1_LOF (0x02) Loss of frame. NMSV5_E1_AIS (0x04) Alarm indication signal. NMSV5_E1_RAI (0x08) Remote alarm indicator. NMSV5_E1_CRC_BLOCK_ERR (0x010) CRC block error. NMSV5_E1_CRC_BLOCK_INFO (0x020) CRC block information (FEBE). NMSV5_E1_NORMAL_FRAMES_0 (0x040) Normal E1 frames (remote SA7 = 0). NMSV5_E1_NORMAL_FRAMES_1 (0x080) Normal E1 frames (remote SA7 = 1).
Slips	DWORD	count	Number of slips.
Es	DWORD	count	Number of errors seconds.
Ses	DWORD	count	Number of severely errored seconds.
Uas	DWORD	count	Number of unavailable seconds.
LineErrors	DWORD	count	Number of line code violations.
FrameErrors	DWORD	count	Number of frame bit errors and CRC errors.
ElaspedTime	DWORD	count	Seconds since counters started. This represents the duration of the observation time (in seconds). **Note:** All statistics are the count of particular events that occur during the ElaspedTime interval.

NMS_V5_CHANNEL_STATISTICS_T

When the NMS GR303 library returns HDLC channel statistics, it provides the information in an NMS_V5_CHANNEL_STATISTICS_T structure (which includes both a CHANNEL_STATISTICS_RECEIVE_T or CHANNEL_STATISTICS_TRANSMIT_T substructure):

```
typedef struct {
   DWORD Size; /* Size of this structure */
   CHANNEL_STATISTICS_RECEIVE_T  chan_stat_rx;
   CHANNEL_STATISTICS_TRANSMIT_T chan_stat_tx;
} NMS_V5_CHANNEL_STATISTICS_T;

typedef struct {
   DWORD Octets;
   DWORD Frames;
   DWORD Drops;
   DWORD FifoOverruns;
   DWORD Aborts;
   DWORD CrcErrors;
   DWORD NonAlignedOctets;
   DWORD BufferOverflows;
} CHANNEL_STATISTICS_RECEIVE_T;

typedef struct {
   DWORD Octets;
   DWORD Frames;
   DWORD Drops;
   DWORD FifoUnderruns;
   DWORD FifoOverruns;
} CHANNEL_STATISTICS_TRANSMIT_T;
```

CHANNEL_STATISTICS_RECEIVE_T

Dependent function: **NMS_V5GetChannelStatistics**

Field name	Type	Units	Description
Octets	DWORD	count	Bytes received count.
Frames	DWORD	count	Frames received count.
Drops	DWORD	count	Frames dropped count.
FifoOverruns	DWORD	count	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.
Aborts	DWORD	count	Receive abort count.
CrcErrors	DWORD	count	Receive CRC errors count.
NonAlignedOctets	DWORD	count	Non-aligned octets count. Not supported on CG 6100C or CG 6500C boards.
BufferOverflows	DWORD	count	Buffer overflows count. Not supported on CG 6100C or CG 6500C boards.

CHANNEL_STATISTICS_TRANSMIT_T

Dependent function: **NMS_V5GetChannelStatistics**

Field name	Type	Units	Description
Octets	DWORD	count	Bytes transmitted count.
Frames	DWORD	count	Frames transmitted count.
Drops	DWORD	count	Frames dropped count.
FifoUnderruns	DWORD	count	FIFO underrun count. Not supported on CG 6100C or CG 6500C boards.
FifoOverruns	DWORD	count	FIFO overrun count. Not supported on CG 6100C or CG 6500C boards.

14 T1 and E1 trunk channels

Channels and transmission rates

T1 and E1 trunks are four-wire digital transmission links. T1 trunks are used mainly in the United States, Canada, Hong Kong, and Japan. E1 trunks are used in Europe.

Data on a T1 or E1 trunk is transmitted in channels. Each channel carries information digitized at 64000 bits per second (bps). This transmission rate is called the digital signal level 0 (DS-0) rate.

T1 trunks carry 24 channels. E1 trunks carry 32 channels. The total throughput rate (called digital signal level 1 or DS-1) is:

- For T1 trunks: 24 channels, each carrying 64,000 bps, yield a throughput rate of 1,536,000 bps. An extra 8000 bps are used to carry framing and other information (as described in *Framing* on page 193). DS-1 for T1 trunks is 1,544,000 bps.
- For E1 trunks: 32 channels, each carrying 64,000 bps, yield a rate of 2,048,000 bps.

Signaling

Two types of information are carried on a trunk:

- Voice information
- Signaling information (indicating, for example, a channel is on-hook or off-hook)

Signaling information can be conveyed using either channel associated signaling (CAS) or common channel signaling (CCS).

Note: The following information is provided for informational use only. Your board's hardware performs all the operations necessary to support the framing system used on the trunk. The TCPs perform all necessary signaling operations.

Channel associated signaling (CAS)

With channel associated signaling (CAS), signaling information is sent for all channels at regular intervals, regardless of whether each channel's state changes. The information for each channel consists of a set of bits called the ABCD bits. Whenever a channel's state changes, the ABCD bit pattern for that channel changes to convey the signaling bits.

On T1 trunks using a CAS protocol, the signaling information for each channel is transmitted using a method called robbed-bit signaling. With this method, one of the bits in the voice information in each channel is changed at regular intervals to indicate the state of the channel. Since the intervals are widely spaced, sound quality in the channel is not compromised.

On E1 trunks using a CAS protocol, channel 16 carries the ABCD bits for all of the other channels. No robbed-bit signaling is used.

Different CAS protocols use the ABCD bits in different ways. For example, some protocols use only two bits to signal four separate states; the other bits are not used. Other protocols convey signaling using one bit only, by setting and resetting the bit at specific intervals to signal different states. The specific patterns of bits used to indicate signaling states differ from country to country. Refer to the appropriate protocol reference manual for more information.

Common channel signaling (CCS)

With common channel signaling (CCS), packets of signaling information for a channel are sent when the channel's state changes, instead of signaling bits. CCS information is sent in a dedicated channel, the data channel or D channel. Voice information is carried in bearer channels (B channels).

Different CCS protocols reserve different channels for transmitting data. For example, GR303 occupies channels 24 and 12 on T1 trunks to run TMC and EOC protocols. GR303 also uses robbed-bit signaling for call control on established connections. On E1 trunks using the V5 protocol, signalling takes place on channels 15, 16, and 31.

Framing

On T1 and E1 trunks, the data in the channels is combined into a single continuous stream of data using time-division multiplexing (TDM). With TDM, the channels take turns sharing the trunk over and over again. Each channel broadcasts 8 bits at a time. The time given a channel during a given round is called a timeslot. One cycle of timeslots is called a frame.

T1 and E1 trunks delineate frames differently. This topic describes:

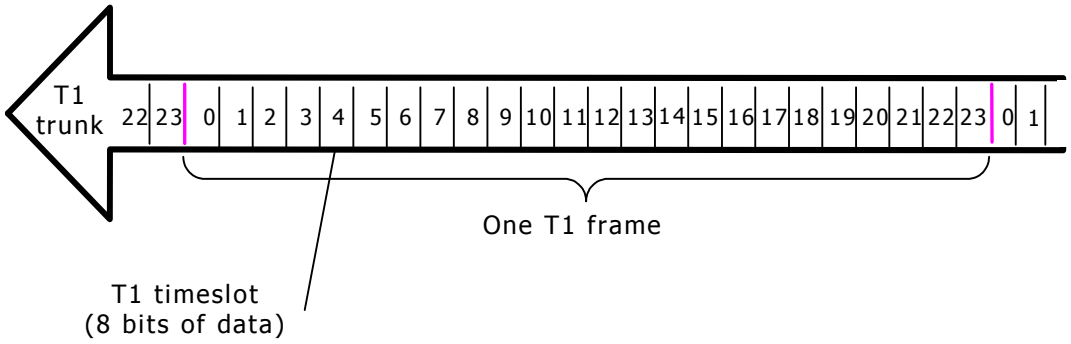
- T1 framing formats
- E1 framing formats

When configuring a CG board, you specify which framing format to use with the NetworkInterface.T1E1[***x***].FrameType keyword. For more information about configuring the CG board, refer to the CG hardware documentation.

Note: The following information is provided for informational use only. Your board's hardware performs all the operations necessary to support the framing system used on the trunk. The TCPs perform all necessary signaling operations.

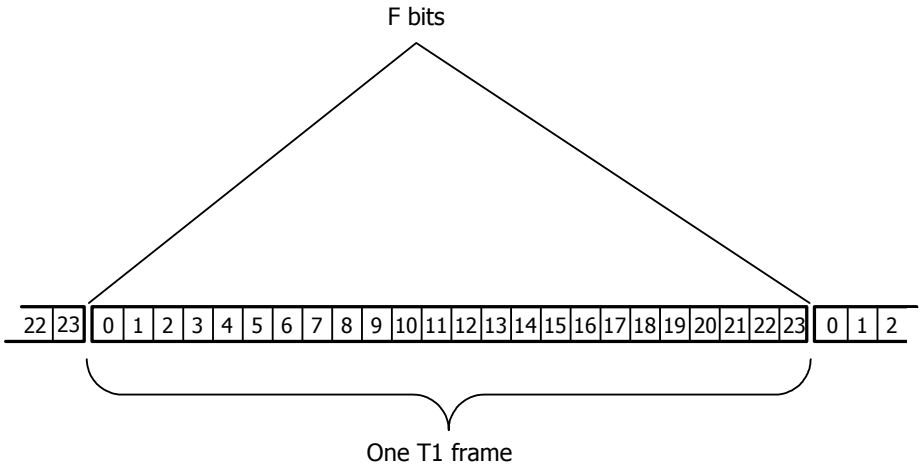
T1 framing formats

On T1 trunks, a frame consists of 24 timeslots, sent every 125 µsec (1/8000 sec).



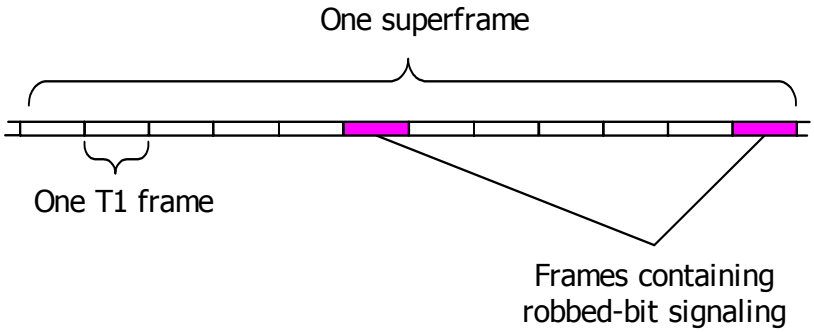
CG boards support two T1 framing formats: D4 framing and Extended SuperFrame (ESF).

With D4 framing, a single framing bit (F bit) is sent after each frame to mark the end of the frame and the beginning of the next one. Each frame consists of (24x8)+1 = 193 bits. The framing bits (8000 per second) take up extra bandwidth. The following illustration shows framing bits on a T1 trunk:



After each frame, the F bit is set or reset according to a pattern that repeats once every 12 frames: 100011011100. This makes the F bit recognizable even in the high-speed T1 bit stream. The 12 frames in this cycle constitute one superframe.

With CAS protocols, the least significant bit in each timeslot is robbed for signaling in the 6th and 12th frames in each superframe. Since each bit has only two possible states (0 or 1), only four separate signaling conditions can be transmitted with CAS protocols. The following illustration shows robbed-bit signaling (D4 framing format):

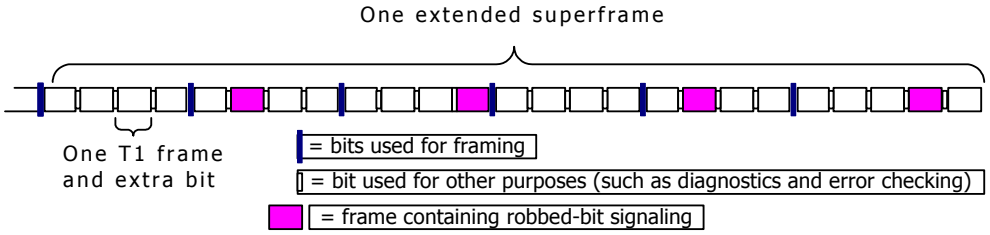


With ESF framing, an extra bit appears after every frame, as in D4 framing. However, only every fourth extra bit is used for framing. This bit is set or reset in a pattern that repeats once every 24 frames, instead of the 12-frame repetition in D4 framing. The 24 frames in the cycle constitute one extended superframe.

All of the other extra bits (18 in all) are used alternately:

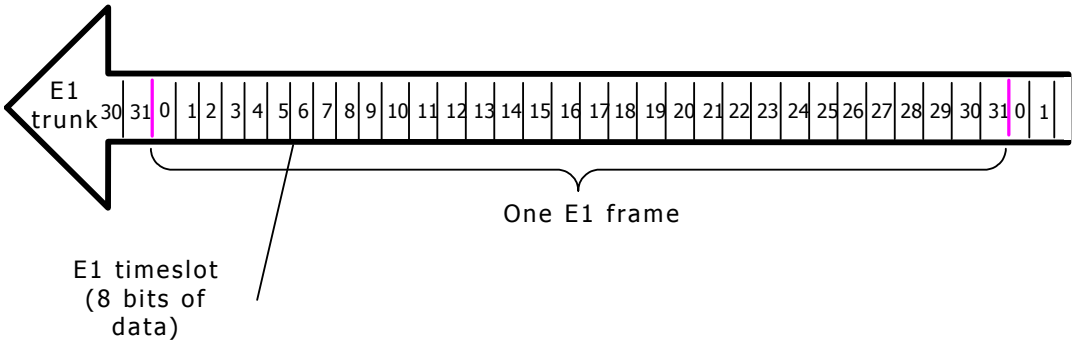
- Six of the bits are used for a cyclic redundancy check (CRC) to detect errors.
- The other 12 carry diagnostic data. This bandwidth is called the facilities data link (FDL).

With CAS protocols, bits are robbed from each timeslot in the 6th, 12th, 18th, and 24th frame in the extended superframe. Thus instead of two signaling bits per superframe, ESF has 4 bits, allowing up to 16 separate signaling conditions to be transmitted. The following illustration shows extended superframe:



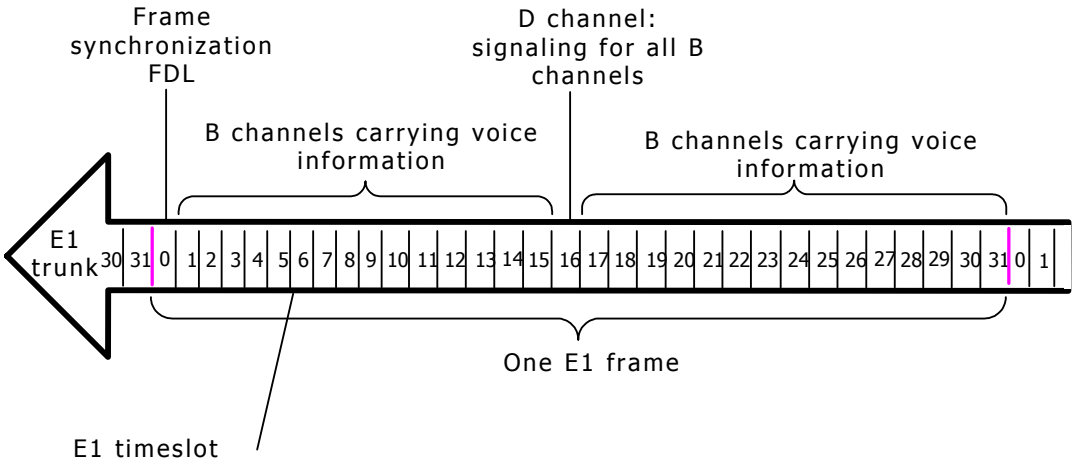
E1 framing formats

On E1 trunks, a frame consists of 32 timeslots. A frame is sent every 125 µsec (1/8000 sec):



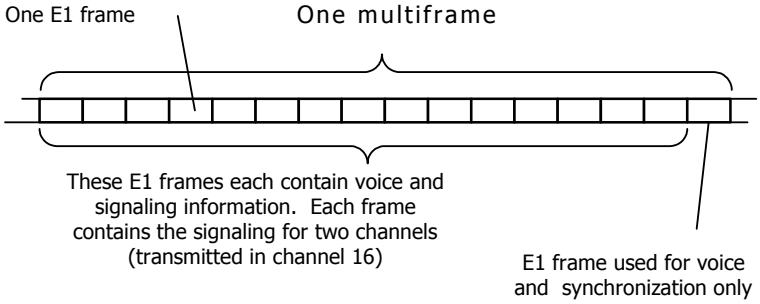
In each frame, channels are numbered 0 through 31. Half of the first channel (channel 0) is used for frame synchronization. The other half can be used as a facilities data link (FDL).

With CAS protocols, signaling information for each channel is carried in channel 16. This eliminates the need for robbed-bit signaling. Channels 1 through 15 and 17 through 31 (30 channels in all) carry voice data information. The following illustration shows CEPT E1 timeslots:



With CAS protocols, four ABCD bits are sent for each channel at a time. Since timeslot 16 can carry only 8 bits of information per frame, it is not possible to send the signaling for all 30 channels in each frame. Therefore, channels take turns using channel 16, two at a time. It takes 15 frames to cycle through the signaling for all channels.

After every 15 frames, an extra frame is sent to synchronize the receiver to the signaling channel. Thus the full cycle contains 16 frames. A group of 16 such frames is called a multiframe. The following illustration shows an E1 multiframe:



Voice encoding

For CG boards, the information received is already pulse code modulation (PCM) encoded.

Only 256 possible amplitude measurements can be represented with 8 bits. 256 digital values are not enough to represent the entire amplitude range of the human voice at a usable quality level. However, most of the characteristics of a voice signal that make it understandable to the human ear exist at the lower end of the amplitude range. Therefore, the values are assigned to amplitude values non-linearly, with many values available to represent various amplitudes in the low end of the range, and few values to measure the high end. This compression method is called companding.

Different companding algorithms are used in different geographic regions. A companding method called mu-law is used in the US, Canada, and Japan. Another method, called A-law, is used in the rest of the world.

When configuring a CG board, you select mu-law or A-law versions of the DSP files.

AMI, ones density, and zero code suppression

To reduce crosstalk on T1 and E1 trunks, and to eliminate DC bias, each 1 bit on the trunk is sent with the opposite electrical polarity of the preceding 1 bit. This transmission method is called alternate mark inversion (AMI).

Zero bits are sent as intervals of zero voltage. Multiple zeros in a row appear at the receiving end as one long interval of no voltage. If these gaps are too long, it is difficult for the receiving end to maintain framing synchronization with the transmitting end. Algorithms used in T1 and E1 transmissions solve this problem by ensuring that sufficient ones (enough ones density) keep the transmitting and receiving ends synchronized. These algorithms, called zero code suppression algorithms, are described in the following table:

Algorithm	Description
B8ZS - binary 8-zero suppression	Used with ISDN protocols. To send an interval of successive zeroes, the sending end replaces the zeroes with a pattern of ones and zeroes in which bipolar violations occur; that is, one or more successive ones are sent with the same polarity, disrupting the AMI pattern. The pattern of bipolar violations is recognized at the receiving end and turned back into zeroes.
HDB3	High density bipolar 3 code uses patterns of bipolar violations to replace sequences of 4 zero data bits in order to maintain ones density on clear channel transmission.
Jammed-bit 7 zero code suppression	In an interval of zeroes, the sending end jams every bit 7 high so the receiving end can recognize it. This method sacrifices data integrity, but quality is sufficient for voice transmissions.
No zero code suppression	Zero code suppression disabled.

CG boards configured as E1 boards can be set up to transmit without zero code suppression, or to use the high density bipolar 3 code (HDB3) algorithm. In HDB3, sequences of 4 zero data bits are replaced by patterns of bipolar violations.

When configuring a CG board, use the NetworkInterface.T1E1[***x***].LineCode keyword to specify which algorithm to use. For more information, refer to the CG board documentation.

Index