Making Innovation Thrive™

Dialogic® Global Call ISDN

Technology Guide

December 2008

05-2242-008

Publication Date: December 2008

Document Number: 05-2242-008

Contents

Figures

Tables

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2242-008	December 2008	Made global changes to reflect Dialogic brand. ISDN-Specific Operations chapter: Added Using Optional ISDN IEs with Dialogic® DM3 Boards. Added Sending and Receiving DPNSS End to End Messages. Debugging Dialogic® Global Call ISDN Applications chapter: Added that the pritrace Utility is supported on Dialogic® DM3 Boards (as well as on Dialogic® Springware Boards). ISDN-Specific Function Information chapter: Under gc_SndMsg( ) Variances for ISDN, added SndMsg_RawEEM to the message types supported when using Dialogic® DM3 Boards. ISDN-Specific Parameter Reference chapter: Under GCIS_SET_DLINK Parameter Set, deleted note that said this parameter set is not supported when using Dialogic® DM3 Boards, which was incorrect. The GCIS_PARM_DLINK_STATE parameter ID is supported on Dialogic® DM3 Boards (as well as on Dialogic® Springware Boards). ISDN-Specific Event Cause Values chapter: Made additions to Firmware-Related Cause Values When Using Dialogic® DM3 Boards table. (IPY00041046)
05-2242-007	May 2006	General: Updates to indicate that TBCT is an NI2 supplementary service supported on 5ESS and DMS switches that implement NI2. General: Removed references to "fcdgen" in multiple places. Dialogic® Global Call API Functions Supported by ISDN section: For **gc_SetInfoElem( )**, updated the "(deprecated)" label to "(deprecated for Springware boards)". Controlling B Channel Status section: Revised text to remove unrelated information about **gc_WaitCall( )** and **gc_ResetLineDev( )**. Retrieving Continuity Check IE section: Added section. Using the GC_MAKECALL_BLK Structure section: Added a caution about the use of the origination_phone_number field in the MAKECALL_BLK structure. gc_SetCallingNum( ) Variances for ISDN section: Added a caution about the use of the origination_phone_number field in the MAKECALL_BLK structure. ISDN Network Firmware section: Removed the note stating that network-side firmware is for test purposes only. Network-side protocols are supported.
05-2242-006	January 2006	Non-Call Associated Signaling on DM3 Boards section: Added. Network Facility Request - Two B Channel Transfer (Synchronous Mode) section: Updated to indicate support for DMS and 5ESS only. Using Overlap Send: Added note to identify supported protocols: Net5 and QSIG. Using Dynamic Trunk Configuration section: Added note to identify the application's responsibilities for terminating calls if necessary and explain the use of gc_ResetLineDev( ) in this context. Setting the ISDN Protocol Mode for a Trunk section: Removed the second note at end of section; now covered by the note added in the parent section.

Document No.	Publication Date	Description of Revisions
05-2242-006 (continued)		Setting the Line Type and Coding for a Trunk section: Removed the statement mandating the termination of calls and the use of gc_ResetLineDev( ); now covered by the note in the parent section. Specifying the Protocol for a Trunk section: Removed the statement mandating the termination of calls and the use of gc_ResetLineDev( ); now covered by the note in the parent section. GCIS_SET_BEARERCHNL Parameter Set section: Updated to reflect support for QSIG NCAS on DM3 boards. GCIS_SET_CALLTYPE Parameter Set section: Added.
05-2242-005	December 2005	Responding to ISDN Events table: Corrected the event type (notification) for the events corresponding to the following ext_id values: GCIS_EXEV_NOTIFY, GCIS_EXEV_NOUSRINFOBUF, GCIS_EXEV_L2FRAME and for the GCEV_L2FRAME event. ISDN Extension IDs table: Rephrased the note under GCIS_EXID_SNDMSG. gc_CallProgress( ) Variances for ISDN section: Removed the incorrect reference to the generic method of call progress analysis. gc_SndMsg( ) Variances for ISDN section: Added the SndMsg_Progress message type in the DM3-specific variances subsection.
05-2242-004	September 2005	Alarm Handling for DM3 Boards: Updated to more accurately specify the alarms that can be transmitted to the remote side and provide a mapping to the 0x1626 parameter in the CONFIG file, which is used for trunk preconditioning. Alarm Handling for Springware Boards: Updated to more accurately specify the alarms that can be transmitted to the remote side. ISDN Trace Capability on Multiple Trunks: Added section to describe tracing on multiple trunks for DMT160TEC and DMN160TEC boards.
05-2242-003	August 2005	How to Use This Publication section: Added missing synopsis of chapter 5; also fixed incorrect chapter numbering. Non-Call Associated Signaling on Springware Boards (Synchronous Mode) section: Indicated explicitly the ISDN protocols for which NCAS is supported. Also, clarified which T1 and E1 channels are used for NCAS calls. Also, added a caution related to routing on the D channel for Springware boards with T1 interfaces. (PTR 35249) Alarm Handling for DM3 Boards section: Added new E1 alarms DTE1_BPVS, DTE1_CECS and DTE1_ECS and T1 alarms DTT1_BPVS, DTT1_ECS, DTT1_FEER and DTT1_OOF for DM3 boards. (FR 1365) Alarm Handling for DM3 Boards section: Removed unsupported alarms: DTE1_DCHAN_CFA, DTE1_DCHAN_CFAOK, DTT1_DCHAN_CFA and DTT1_DCHAN_CFAOK. (PTR 34320) Setting the Line Type and Coding for a Trunk section: Added. Specifying the Protocol for a Trunk section: Added. Using Overlap Send section: Deleted paragraph and example indicating that gc_SendMoreInfo( ) is not supported and how to use gc_SndMsg( ) to do overlap send. gc_SendMoreInfo( ) is supported and is the recommended way of doing overlap send. (PTR 34497) ISDN Call Setup Parameters table: Updated the list of supported parameters and values for DM3 boards. (PTR 35521) Global Call Functions Supported by ISDN section: Removed the “deprecated” label next to the gc_SetParm( ) function.

Document No.	Publication Date	Description of Revisions
05-2242-003 (continued)		Global Call Functions Supported by ISDN section: Added new supported utility functions: gc_util_copy_parm_blk( ), gc_util_find_parm_ex( ), gc_util_insert_parm_ref_ex( ) and gc_util_next_parm_ex( ) and new unsupported functions: gc_AcceptModifyCall( ), gc_RejectModifyCall( ), gc_ReqModifyCall( ) and gc_SetAuthenticationInfo( ). gc_AnswerCall( ) Variances for ISDN section: Updated the “Springware-specific Variances” subsection to indicate that a gc_DropCall( ), gc_ReleaseCallEx( ) combination should be used (rather than gc_**ResetLineDev( )**) to recover from the glare condition described. (PTR 35844)
05-2242-002	November 2004	ISDN-Specific Extension IDs section: Updates to clarify the difference between GCEV_EXTENSIONCMPLT and GCEV_EXTENSION. GCEV_EXTENSION Events section: added to describe GCEV_EXTENSION event usabe for Springware and DM3. Using Dynamic Trunk Configuration section: Added information on dynamically configuring a trunk. Network Facility Request - Two B Channel Transfer (Synchronous Mode) section: Added text to clarify that the example code applies to Springware boards. Non-Call Associated Signaling on Springware Boards (Synchronous Mode) section: Updates to indicate all ISDN protocols supported and to explicitly identify the channels used for NCAS (PTR 32165) Implementing Call Hold and Retrieve: Added PRI NTT to the list of protocols that support hold and retrieve on Springware boards. Using Non-Facility Associated Signaling (NFAS): New section. ISDN Network Firmware section: Added note on restriction relating to back-to-back testing on DM3 boards (PTR 33077). gc_AcceptCall( ) Variances for ISDN and gc_AnswerCall( ) Variances for ISDN: Updates for greater consistency between sections. Removed statement indicating that the “rings” parameter is not supported. gc_AnswerCall( ) Variances for ISDN: Updates for consistency. gc_GetNetCRV( ) Variances for ISDN: Deleted section (PTR 32418) Using the gc_SetInfoElem( ) Function section: Updated code example. gc_OpenEx( ) Variances for ISDN section: Updates to address new dynamic trunk configuration capabilities. pritrace Utility section: New section (PTR 27398)
05-2242-001	November 2003	Initial version of document. Much of the information contained in this document was previously published in the *Global Call ISDN Technology User's Guide*, document number 05-0653-008. Major changes since this document version are listed below. General: Updates to indicate that when using **gc_OpenEx( )** with DM3 boards, a voice device can now be specified in the **devicename** string. Default Channel States for DM3 and Springware Boards section: Added section to describe default channel states following firmware download (PTR 25482) Responding to ISDN Events table: Updated text descriptions for call hold and retrieve events to indicate support when using DM3 boards. Responding to ISDN Events table: For GCEV_FACILITY (Springware) and GCEV_EXTENSION with id of GCIS_EXEV_FACILITY (DM3) changed function used to retrieve information to gc_GetSigInfo( ) instead of gc_GetCallInfo( ).

Document No.	Publication Date	Description of Revisions
05-2242-001 (continued)		Call Progress Analysis When Using DM3 Boards section: Added a reference to the Global Call API Programming Guide that describes a new method of Call Progress Analysis (CPA). Also added a subsection to reference the CPA parameter defaults in the CONFIG file and to indicate that a voice device can now be specified when issuing gc_OpenEx( ). Implementing Call Hold and Retrieve section: Added section to describe the functions used to implement call hold and retrieve and the level of support provided when using DM3 and Springware boards. Using Dynamic Trunk Configuration section: Added section for dynamic trunk configuration on DM3 boards. Set ISDN-Specific Event Masks section: Deleted GCISMSK_TERMINATE from the list of supported masks in the GC_PARM_BLK. (P/O PTR 29203) ISDN Network Firmware section: Added a note to clarify that ISDN Network Firmware is provided for back-to-back testing purposes. (PTR 30475) Network-Terminated Call When the Application Does Not Drop the Call section: Describes a scenario where there are two simultaneously active CRNs when the application does not issue gc_DropCall( ) to release the first call before a second call arrives. Call Hold and Retrieve Scenarios section: Added section to describe scenarios for DM3 boards. Alarm Handling for DM3 Boards section: Removed DTE1_CRC_CFA (time slot 16 CRC failure) and DTE1_CRC_CFAOK (time slot 16 CRC failure recovery) from the list of supported alarms when using ISDN on E1 interfaces. Handling Errors section: Created separate sections describing ISDN cause codes for DM3 and Springware and added more specific DM3 information. gc_SetChanState( ) Variances for ISDN section: Fixed note that indicated DM3 was not supported. gc_SetConfigData( ) Variances for ISDN section: Updated to indicate support for dynamic trunk configuration on DM3 boards. gc_SetEvtMsk( ) Variances for ISDN section: Updated to better reflect DM3 and Springware functionality. gc_SetInfoElem( ) Variances for ISDN section: Removed the note stating that gc_SetInfoElem( ) is not supported when using DM3 board. The function is supported when using DM3 boards. (P/O PTR 29204) gc_SetUserInfo( ) Variances for ISDN section: Added note to indicate that gc_SetUserInfo( ) is not supported when using DM3 boards. (PTR 29204) gc_SndMsg( ) Variances for ISDN section: Updated to indicate that this function is not deprecated when using DM3 boards. GCIS_SET_EVENTMSK Parameter Set section: Deleted GCISMSK_TERMINATE from the set of valid values for the three parameters in the GCIS_SET_EVENTMSK parameter set. (P/O PTR 29203) ISDN-Specific Event Cause Values chapter: Added call control library-related and firmware-related cause code values for DM3. : B Channel Negotiation section: Added section to describe support for B channel negotiation for PRI protocols. Global Call Functions Supported by ISDN section: Added unsupported new call transfer functions. gc_GetNetCRV( ) Variances for ISDN section: Added note to indicate that setting the NetCRV Support parameter is not supported for DPNSS and DASS2 protocols and must be set to 0. (PTR 31410)

Document No.	Publication Date	Description of Revisions
05-2242-001 (continued)		gc_OpenEx( ) Variances for ISDN section: Added information about differences at the firmware level between Springware and DM3 and how this translates at the Global Call level. (PTR 29177) gc_HoldACK( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3. gc_HoldCall( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3. Also added text to indicate GCEV_HOLDREJ received if gc_HoldCall( ) issued before the Connected state. (PTR 30930) gc_HoldRej( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3. gc_MakeCall( ) Variances for ISDN section: Changed text describing the maximum number of digits in the numberstr parameter. (PTR 22842) gc_RetrieveAck( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3. gc_RetrieveCall( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3. gc_RetrieveRej( ) Variances for ISDN section: Changed the note to indicate that the function is fully supported on DM3.

About This Publication

The following topics provide information about this publication.

- Purpose
- Applicability
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This guide is for users of the Dialogic® Global Call API who choose to write applications that use Integrated Services Digital Network (ISDN) technology. This guide provides Global Call ISDN-specific information only and should be used in conjunction with the *Dialogic® Global Call API Programming Guide* and the *Dialogic® Global Call API Library Reference*, which describe the generic behavior of the Global Call API.

Applicability

This document version is applicable to Dialogic® Host Media Processing (HMP) Software and to Dialogic® System Release Software for Linux and Windows® operating systems.

Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This guide is intended for:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

This publication assumes that the audience is familiar with the Windows® and Linux operating systems and has experience using the C programming language.

How to Use This Publication

Refer to this guide if you have installed the system software that includes the Global Call software.

This guide is divided into the following chapters:

- Chapter 1, "ISDN Overview" gives a brief introduction to ISDN technology for novice users.
- Chapter 2, "Dialogic® Global Call Architecture for ISDN" describes how the Global Call API can be used with ISDN technology and provides an overview of the architecture.
- Chapter 3, "ISDN Call Scenarios" provides some call scenarios that are specific to ISDN technology.
- Chapter 4, "ISDN-Specific Operations" describes how to use the Global Call API to perform ISDN-specific operations, such sending a Progress message to the network, retrieving D channel status, overlap sending etc.
- Chapter 5, "ISDN Protocols" describes the ISDN protocols supported by the Global Call API, the firmware and parameter files for each protocol, and protocol parameters.
- Chapter 6, "Building Dialogic® Global Call ISDN Applications" provides guidelines for those choosing to develop Global Call applications that use ISDN technology.
- Chapter 7, "Debugging Dialogic® Global Call ISDN Applications" provides information for debugging Global Call applications that use ISDN technology.
- Chapter 8, "ISDN-Specific Function Information" describes the additional functionality of specific Global Call functions used with ISDN technology.
- Chapter 9, "ISDN-Specific Parameter Reference" provides a reference for ISDN-specific parameter set IDs and their associated parameter IDs.
- Chapter 10, "ISDN-Specific Data Structures" provides a data structure reference for ISDN-specific data structures.
- Chapter 11, "ISDN-Specific Event Cause Values" provides descriptions of ISDN-specific event cause codes.
- Chapter 12, "Supplementary Reference Information" provides supplementary information including technology references and IE and message type formats for DPNSS.
- A Glossary and an Index can be found at the end of the document.

Related Information

See the following for additional information:

- *http://www.dialogic.com/manuals/* (for Dialogic® product documentation)
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

ISDN Overview 1

This chapter provides a brief overview of Integrated Services Digital Network (ISDN) technology. This is a high-level description of the technology and does not provide details of any aspect of ISDN technology. Some references to where more detailed information can be obtained are provided.

Topics covered by this chapter include:

1.1 ISDN Definition

ISDN is a collection of internationally accepted standards for defining interfaces and operation of digital switching equipment for the transmission of voice, data, and signaling. ISDN has the following characteristics:

- ISDN makes all transmission circuits end-to-end digital.
- ISDN adopts a standard out-of-band signaling system.
- ISDN brings significantly more bandwidth to the desktop.

1.2 ISDN Features and Benefits

ISDN is a digital communications network capable of carrying all forms of digitized data (voice, computer, and facsimile) between switched end points. This network is a digital-switched system that makes a connection only when requested.

Control over switched connections is provided by a protocol of messages that pass between the two ends of the digital link. Any type of equipment can be connected to an ISDN, provided the equipment is capable of generating a digital bit stream that conforms to ISDN standards.

ISDN technology offers the benefits inherent in digital connectivity such as fast connection (setup and tear-down), fast Direct Dialing In service (DDI), and fast Automatic Number Identification (ANI) acquisition. In addition, ISDN Primary Rate Interface (PRI) applications can take advantage of the following features, if offered by the network (see Section 3.1, "General ISDN Call Scenarios", on page 47, for details):

Two B Channel Transfer (TBCT)
: TBCT is a National ISDN-2 (NI2) supplementary service described in the Telcordia GR 2865 standard. The feature enables a user to request the switch to connect together two independent calls on the user's interface. The user who made the request is released from the calls and the other two users are directly connected. This feature is supported on 5ESS and DMS switches provisioned to implement NI2; see Section 3.1, "General ISDN Call Scenarios", on page 47 for details.

Non-Call Associated Signaling (NCAS)
: Allows users to communicate via user-to-user signaling without setting up a circuit-switched connection (this signaling does not occupy B channel bandwidth). A temporary signaling connection is established (and cleared) in a manner similar to the control of a circuit-switched connection. This feature is supported for the 5ESS protocol. For details, see Section 3.1, "General ISDN Call Scenarios", on page 47.

Vari-A-Bill
: A flexible billing option enabling a customer to modify the charge for a call while the call is in a stable state (for example, between answer and disconnect). This feature is available from the AT&T network only.

ANI-on-demand
: Allows the user to request a caller ID number to identify the origin of the call, when necessary. Applies to AT&T only.

Non-Facility Associated Signaling (NFAS)
: Provides support for multiple ISDN spans from a single D channel. See the Release Guide for your Dialogic® Software Release for the products that support the NFAS D channel.

Direct Dialing In (DDI)
: A service, also called Dialed Number Identification Service (DNIS), that allows an outside caller to dial an extension within a company without requiring an operator's assistance to transfer the call.

User-to-User Information
: The ability to include an information element (IE) in setup, connect, or disconnect messages.

Call-by-Call service selection
: This feature allows the user to access different services, such as an 800 line or a WATS line, on a per call basis.

LAP-D Layer 2 Access
: Known as the data link layer, this feature provides reliable transfer of data across the physical link and sends blocks of frames with the necessary synchronization, error control, and flow control.

1.3 ISDN Signaling Concepts

This section provides high-level information about ISDN signaling. Topics include:

- Signaling Overview
- Framing
- Data Link Layer Frames
- Network Layer Frames

1.3.1 Signaling Overview

ISDN protocols use an out-of-band signaling method, carrying signaling data on a channel or channels separate from user data channels. This means that one signaling channel (D channel) carries signaling data for more than one bearer channel (B channel). This signaling technique is referred to as common channel signaling (CCS). Signaling data carries information such as the current state of the channel (for example, whether the telephone is on-hook or off-hook). Common channel signaling allows the transmission of additional information, such as ANI and DNIS digits, over the signaling channel.

An ISDN Primary Rate Interface (PRI) trunk provides a digital link that carries some number of TDM (Time Division Multiplexed) channels:

- a T1 trunk carries 24, 64 Kbit channels – 23 voice/data channels (B channels) and one signaling channel (D channel), on a single 1.544 MHz digital link
- an E1 trunk carries 32, 64 Kbit channels – 30 voice/data channels and two additional channels: one signaling channel (D channel) and one framing channel to handle synchronization, on a single 2.048 MHz digital link

The ISDN digital data stream contains two kinds of information: user data and signaling data used to control the communication process. For example, in telephony applications user data is digitally encoded voice data. Voice data from each time slot is routed to a separate B channel. Signaling data carries information such as the current state of the channel (for example, whether the telephone is on-hook or off-hook). The signaling information for all B channel information is routed to the D channel of the device.

The primary rate implementations provided by the Dialogic® Global Call Software comply with most switch protocols worldwide. For the most up-to-date list of available protocols, contact your nearest Sales Office or visit our web site.

1.3.2 Framing

A single frame contains information from each of the B channels and from the D channel, providing a “snapshot” of the data being transmitted at any given time. A frame can be in one of several formats. The frames contain eight bits of information about each time slot or channel. Different frame formats are supported in different networks to provide a variety of added features or benefits.

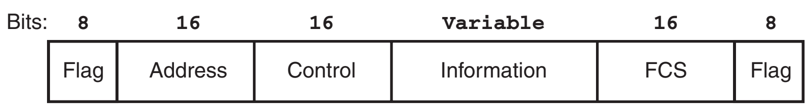
The following frame formats are supported by Dialogic® Global Call ISDN products:

- ESF frame (Extended Superframe)
- D4 frame (Superframe)
- CEPT multiframe (with or without CRC4)

1.3.3 Data Link Layer Frames

The frames that are transmitted over the Data Link Layer (Layer 2) contain information that controls the setup, maintenance, and disconnection between the two physically connected devices as shown in Figure 1.

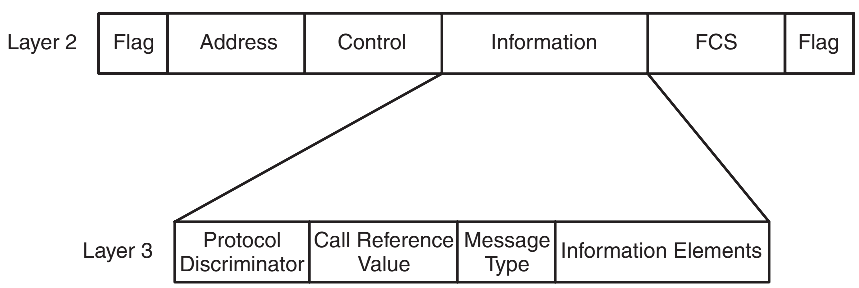
Figure 1. Layer 2 Frame (D Channel)



1.3.4 Network Layer Frames

The Data Link Layer prepares the way for the transmission of Network Layer (Layer 3) frames of data as shown in Figure 2.

Figure 2. Layer 3 Frame (D Channel)



In general, the message format for Layer 3 frames comprises variable length fields with the following format:

Protocol discriminator
: Identifies the protocol type used to handle Layer 3 messages

Call Reference Value (CRV)
: A valued assigned to a call, by the network, for the duration of the call

Message type
: The set of messages used for establishing, controlling, and tearing down a call

Information elements (IEs)
: Used with the message to provide additional information on the type and requirements of the call

1.4 Comparison of ISDN and Analog Connections

ISDN messages can be thought of as a digital equivalent to the analog signaling used to communicate status and connection information across an analog network. Establishing ISDN connections can be related to establishing analog connections as described in Table 1.

Table 1. Comparison of ISDN and Analog Connections

Step	ISDN Connection	Analog Connection
1	The calling party decides to make a call. (See Note below.)	The calling party “goes off-hook.”
2	The calling party sends digital address information to the local Central Office (CO). **Note:** Steps 1 and 2 are the equivalent of the ISDN setup message.	The calling party “dials” the called party’s phone number.
3	The CO accepts the digital address and interconnects local and long-distance circuits, on demand, to reach the called party.	The CO receives the dialed digits and attempts to connect to the called party.
4	The called party receives this address information and responds by sending the calling party an Alerting or Progress message.	The calling party receives either “ringback” or “busy” signal.
5	If the called party accepts the call, a Connect message is sent to the calling party and the parties are connected.	The called party “goes off-hook” to answer the call and the parties are connected.

Many ISDN calls are digital from end-to-end, but a majority are still analog at the ends of the connections. That is, one end or the other connects to a Plain Old analog Telephone Service (POTS). In addition, the call may be routed over both digital and analog links. In these cases, in-band signaling techniques can be used in addition to ISDN signaling so that an application can obtain good feedback from the network regardless of the type of intermediate connections.

Call progress using audio tones is generally not used for digital protocols. The called party’s condition is reported using signaling instead of call progress tones. However, call progress tone detection is desirable for digital circuits for protocols that do not have the capability to report call progress using signaling and when the connection traverses analog lines. For example:

- When a CO is in the telephone path and it cannot transmit the called party’s condition, the busy tone is the only way to recognize a busy condition.
- For telephone circuits that include analog links, the local line may not have access to all of the digital signaling information.

To use call progress in this manner, use the call progress feature in the Dialogic® Voice API after issuing the **gc_MakeCall( )** function. See also Section 2.5, “Resource Association and System Configurations”, on page 35.

1.5 Establishing ISDN Connections

This section provides information about ordering ISDN Primary Rate service and establishing a connection between the Dialogic® Digital Network Interface Boards and the Network Termination Unit (NTU).

Topics include:

- Ordering Service
- Establishing Connections to a NTU

1.5.1 Ordering Service

If ordering your ISDN service from a carrier, keep the following points in mind when talking to a service representative:

- Be specific when describing the kinds of service options you want. Your carrier may offer options that the representative did not mention.
- Find out as much as you can about the setup and connection (turn-up) process.
- Be sure to find out which aspects of service your carrier is responsible for and which aspects are your responsibility. Carriers may offer end-to-end coverage, or responsibility for the lines may lie with several different companies. Not knowing who to contact in the case of difficulties can delay repairs and impact productivity.
- For your customer-site equipment, have available: the manufacturer's name, equipment numbers, and equipment registration numbers for each piece of equipment.

Consider hiring a third party telecommunications or telephone consultant to coordinate service with a carrier. Also, consider delegating parts of the service acquisition process to others. Although these options may involve additional costs, the installation process is streamlined by enlisting the help of someone knowledgeable about the service ordering procedure.

1.5.2 Establishing Connections to a NTU

The Network Termination Unit (NTU) is usually the first piece of equipment on the customer premises that connects to the ISDN line. Customer equipment must be cabled to the NTU. Dialogic does not supply a board-to-NTU cable. You must either purchase one from your supplier or build one yourself. If you are building your own cable, it must fit the following specifications:

Characteristic	Recommendation or Requirement
Cable Type	The recommended cable type is twisted-pair cable in which each of the two pairs is shielded and the two pairs have a common shield as well. Shielding helps prevent noise and the twisting helps prevent crosstalk.
Connectors	The cable connects to the board via an ISO8877 Modular connector on the front or rear bracket of the board. See your NTU documentation for more information.

When building your board-to-NTU cable, be sure you understand how the NTU documentation has labeled NTU pinouts for transmit and receive to local equipment.

Be sure to test your cable after you have built and installed it. The green LEDs on the rear of the Dialogic® Digital Network Interface Board bracket turn on when the board firmware has been downloaded and the board is receiving clocking and synchronization information from the network.

Note: If the pinout appears correct but you receive a red and green light, the transmit and receive may have to be switched on one end.

Dialogic® Global Call Architecture for ISDN 2

This chapter describes the Dialogic® Global Call Software architecture when using ISDN technology and provides a high-level description of how Global Call can be used to develop call control applications that use ISDN. Topics include:

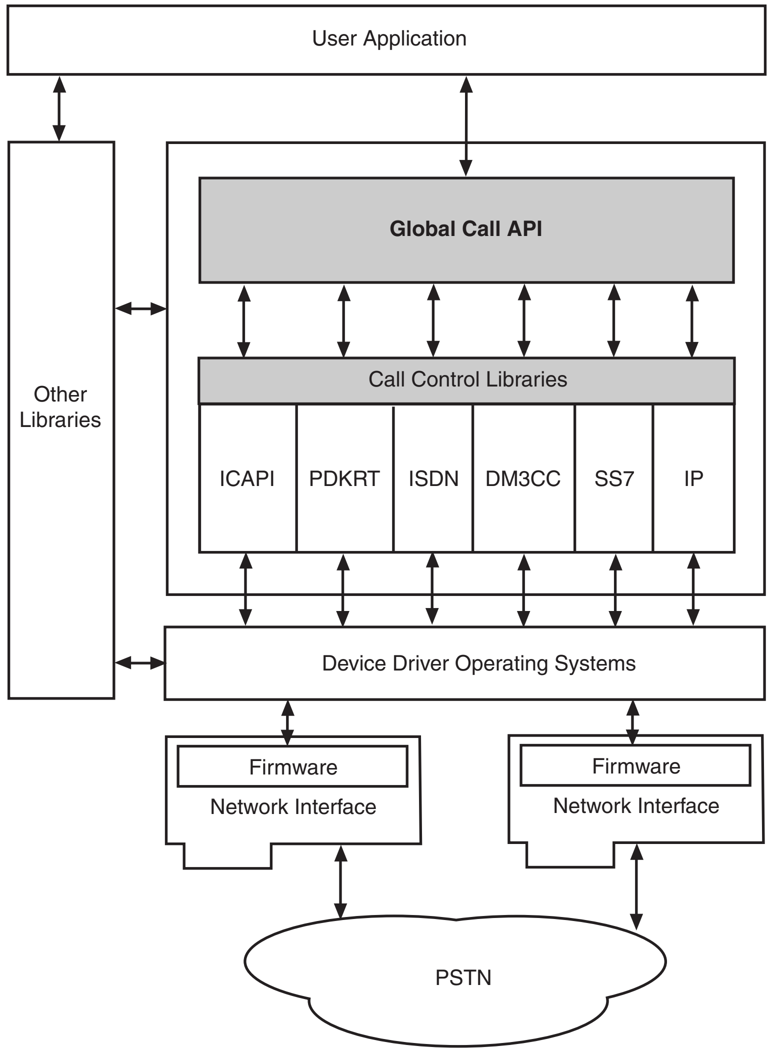
2.1 Dialogic® Global Call Software Architecture When Using ISDN

Figure 3 shows the Dialogic® Global Call Software architecture with the two key elements from an ISDN viewpoint highlighted:

- The Dialogic® Global Call API is a library of functions that provide primarily call control, but also operation and maintenance functionality to applications.
- The underlying ISDN call control library provides the interface between the network and the Dialogic® Global Call library.

See the *Dialogic® Global Call API Programming Guide* for more information on the Dialogic® Global Call Software architecture.

Figure 3. Global Call Architecture When Using ISDN



2.2 Default Channel States for Dialogic® DM3 and Springware Boards

When using Dialogic® DM3 Boards, following firmware download, by default the data link channel (D channel) is in a DOWN state and all bearer channels (B channels) are "out of service". When **gc_OpenEx( )** is executed on a device, the firmware attempts to bring up the D channel and place the B channel associated with the device "in service". If the firmware succeeds, the B channel is placed in the Idle state and can be used for call control. When the application uses **gc_Close( )** to close the B channel, the B channel returns to "out of service".

When using Dialogic® Springware PRI Boards, following firmware download, by default the data link channel (D channel) is in the UP state, assuming there are no blocking alarms on the trunk, and all bearer channels (B channels) are "in service". When using Dialogic® Springware BRI Boards, the D channel must be explicitly put in the UP state using a call to the **cc_SetDChanCfg( )** function (a call control library function).

2.3 Handling ISDN Calls in Asynchronous Mode

The following topics describe the Dialogic® Global Call API functions and events used when processing ISDN calls in asynchronous mode:

- ISDN Inbound Calls in Asynchronous Mode
- ISDN Outbound Calls in Asynchronous Mode
- ISDN Call Termination in Asynchronous Mode

2.3.1 ISDN Inbound Calls in Asynchronous Mode

Table 2 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode inbound call. The items denoted by the dagger symbol (†) are optional functions/events. To prevent interruptions by events that the application does not want to respond to, some events can be masked.

Table 2. ISDN Inbound Call Setup in Asynchronous Mode

Function/Event	Action/Description
gc_WaitCall( )	Issued once after line device opened with **gc_OpenEx( )**. Incoming calls are unblocked.
GCEV_OFFERED	Indicates arrival of an incoming call. A Setup message was received from the network. Proceeding message sent to network: • When using Dialogic® Springware Boards, by default, the Proceeding message is automatically sent to the network. • When using Dialogic® DM3 Boards, by default, the application must explicitly use the **gc_CallAck( )** function to send the Proceeding message. **Note:** The application may connect a voice resource channel to the B channel at this time.
gc_GetDNIS( )†	Request DNIS information. Information returned is stored in a buffer.
gc_GetANI( )† (when using Dialogic® Springware or DM3 Boards) OR **gc_ReqANI( )**† (when using Dialogic® Springware Boards only)	Information returned is stored in a buffer. Request ANI information.
gc_CallProgress( )† (when using Dialogic® Springware Boards only)	Progress message sent to acknowledge that the call was received. No response expected from network.
† indicates an optional function or event.	

Table 2. ISDN Inbound Call Setup in Asynchronous Mode (Continued)

Function/Event	Action/Description
GCEV_CALLPROGRESS†	Termination event
gc_AcceptCall( )†	Alerting message sent to acknowledge that call was received but called party has not answered.
GCEV_ACCEPT†	Termination event - indicates call received, but not yet answered.
gc_AnswerCall( )	**Note:** Application may connect a voice resource channel to the B channel. Connect message sent to connect call to called party (answer inbound call). Calling party may respond with a Connect Acknowledged message.
GCEV_ANSWERED	Termination event - indicates inbound call connected. Causes transition to Connected state.
† indicates an optional function or event.	

2.3.2 ISDN Outbound Calls in Asynchronous Mode

Table 3 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode outbound call. The items denoted by the dagger symbol (†) are optional events that may be reported to the application for specific signaling protocols.

Table 3. ISDN Outbound Call in Asynchronous Mode

Function/Event	Action/Description
gc_MakeCall( )	Requests a connection using a specified line device; a CRN is assigned and returned immediately. Setup message is sent to network.
GCEV_PROCEEDING†	Event indicates that a Proceeding message was received from the network.
GCEV_PROGRESSING†	Event indicates that Progress message was received from network. Multiple events of this type may be received within a call. The application may assign a voice resource to detect the in-band tones.
GCEV_ALERTING†	Event indicates that an Alerting message was received from network indicating that the remote end was reached but a connection has not been established.
GCEV_CONNECTED	Event indicates that a Connect message was received from network. Indicates successful completion of **gc_MakeCall( )**.
† indicates an optional event.	

2.3.3 ISDN Call Termination in Asynchronous Mode

Table 4 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode call termination.

Table 4. Call Termination in Asynchronous Mode

Function/Event	Action/Description
	Disconnect message received when call is terminated by network.
GCEV_DISCONNECTED	Unsolicited event generated when call is terminated by network; initiates transition to Disconnected state.
gc_DropCall( )	Disconnects call specified by CRN.
GCEV_DROPCALL	Termination event - signals that call is disconnected and initiates transition to Idle state.
gc_ReleaseCall( )	Issued to release all resources used for call; network port is ready to receive next call. Causes transition to Null state.

2.4 Handling ISDN Calls in Synchronous Mode

The following topics describe the Dialogic® Global Call API functions and events used when processing ISDN calls in synchronous mode:

- ISDN Inbound Calls in Synchronous Mode
- ISDN Outbound Calls in Synchronous Mode
- ISDN Call Termination in Synchronous Mode

2.4.1 ISDN Inbound Calls in Synchronous Mode

Table 5 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode inbound call. The items denoted by the dagger symbol (†) are optional functions/events or maskable events that may be reported to the application for specific signaling protocols.

Table 5. ISDN Inbound Call Setup in Synchronous Mode

Function/Event	Action/Description
gc_WaitCall( )	Enables notification of an incoming call after line device opened with **gc_Open( )** or **gc_OpenEx( )**. Incoming calls are unblocked.
Incoming call	A Setup message is received from the network. A Proceeding message is sent to network: • When using Dialogic® Springware Boards, by default, the Proceeding message is automatically sent to the network. • When using Dialogic® DM3 Boards, by default, the application must explicitly use the **gc_CallAck( )** function to send the Proceeding message. Application may connect a voice resource channel to the B channel at this time.
gc_GetDNIS( )	Request DNIS information; information returned is stored in buffer.
† indicates an optional function or event.	

Table 5. ISDN Inbound Call Setup in Synchronous Mode (Continued)

Function/Event	Action/Description
gc_GetANI( )†	Information returned is stored in buffer.
gc_CallProgress( )†	Progress message sent to acknowledge that call was received. No response expected from network.
gc_AcceptCall( )†	Alerting message sent to acknowledge that a call was received but called party has not answered.
gc_AnswerCall( )	Application may connect a voice resource channel to the B channel. Connect message sent to connect call to called party (answer inbound call). Calling party may respond with a Connect Acknowledged message.
† indicates an optional function or event.	

2.4.2 ISDN Outbound Calls in Synchronous Mode

See Table 6 for the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode outbound call. The items denoted by the dagger symbol (†) are optional events that may be reported to the application for specific signaling protocols.

Note: When using the synchronous programming model, the application must handle unsolicited events unless the events are masked or disabled. Refer to the **gc_SetEvtMsk( )** function description in the *Dialogic® Global Call API Library Reference* for a list of maskable events.

Table 6. ISDN Outbound Call in Synchronous Mode

Function/Event	Action/Description
gc_MakeCall( )	Requests a connection using a specified line device; a CRN is assigned and returned immediately. Setup is message sent to network.
GCEV_PROCEEDING†	Event indicates that a Proceeding message was received from the network.
GCEV_PROGRESSING†	Event indicates that a Progress message was received from network. Multiple events of this type may be received within a call. The application may assign a voice resource to detect the in-band tones.
GCEV_ALERTING†	Event indicates that an Alerting message was received from network indicating that the remote end was reached but a connection has not been established. When the call is answered, **gc_MakeCall( )** returns.
Completion of **gc_MakeCall( )**	Connect message was received from network.
† indicates an optional event.	

2.4.3 ISDN Call Termination in Synchronous Mode

Table 7 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode call termination.

Table 7. Call Termination in Synchronous Mode

Function/Event	Action/Description
	Disconnect message received when call is terminated by network.
GCEV_DISCONNECTED	Unsolicited event - generated when a call is terminated by the network; initiates transition to Disconnected state. Release message is sent to network. Network responds with Release Complete message.
gc_DropCall( )	Disconnects call specified by CRN.
gc_ReleaseCall( )	Issued to release all resources used for a call; network port is ready to receive the next call. Causes transition to Null state.

2.5 Resource Association and System Configurations

Typically, in ISDN environments, calls do not require voice resources for ISDN signaling. However, voice resources may be used when the call is not end-to-end ISDN and in-band signaling information is to be collected.

Using Dialogic® Global Call ISDN products, applications can control Primary Rate line connectivity. The Dialogic® Global Call ISDN Boards can be configured as terminating devices or installed in a variety of drop-and-insert configurations.

In a terminating configuration, incoming or outgoing calls on ISDN lines are processed by supported resource boards (such as voice boards). In a drop-and-insert configuration, incoming and outgoing calls (on individual channels) can either be processed by supported resource boards or passed on to additional network boards. Calls can also be both processed by supported resource boards and passed on to additional network boards, as well.

Dialogic® Global Call ISDN products can be placed in a variety of drop-and-insert configurations, providing all the features and benefits of terminating configurations, plus the ability to access an operator or another call. Drop-and-insert configurations allow calls to be passed from one network module (such as the Dialogic® DTI/240SC Board) to another network module.

For each call, whether an inbound or an outbound call, the entity making the call is the "calling party" and the entity receiving the call is the "called party". For an inbound call, the calling party is eventually connected to a central office (CO) that connects to the Customer Premises Equipment (CPE) of the called party.

2.6 Responding to ISDN Events

The receipt of an ISDN message or event may require taking the action described in Table 8 to retrieve information or to set up the channel for the next call. The following descriptions supplement the event descriptions listed in the *Dialogic® Global Call API Library Reference*.

Table 8. Responding to ISDN Events

Event	Description/Action
GCEV_CALLINFO when using both Dialogic® Springware and DM3 Boards	Unsolicited ISDN event (not maskable) generated when an incoming Information message is received. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_CONGESTION when using Dialogic® Springware Boards GCEV_CONGESTION event when using Dialogic® DM3 Boards	Unsolicited ISDN event (not maskable) generated when an incoming Congestion message is received indicating that the remote end is not ready to accept inbound user information. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_D_CHAN_STATUS when using both Dialogic® Springware and DM3 Boards	Unsolicited ISDN event (not maskable) generated when the status of the D channel changes as a result of an event on the D channel. Use **gc_GetLineDevState( )** function to retrieve D channel status. Use **gc_ResultInfo( )** function to retrieve a cause code and a description of the cause.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_DIVERTED when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event generated when a NAM with divert information is received. Indicates that an outbound call was successfully diverted to another station (DPNSS protocol only). Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_FACILITY when using Dialogic® Springware Boards GCEV_FACILITY event when using Dialogic® DM3 Boards	Unsolicited ISDN event (not maskable) generated when an incoming Facility Request message is received. Use **gc_GetSigInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_FACILITY_ACK when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (not maskable) generated when an incoming FACILITY_ACKNOWLEDGEMENT message is received. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_FACILITY_REJ when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (not maskable) generated when an incoming FACILITY_REJECT message is received. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_HOLDACK when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Termination event for ISDN **gc_HoldCall( )** function generated when a Hold Call request is acknowledged successfully.
GCEV_HOLDCALL when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Unsolicited event (not maskable) generated when the Hold Call request was acknowledged by the remote end and the call is in the Hold state. Respond with a **gc_HoldAck( )** or **gc_HoldRej( )** function.

Table 8. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_HOLDREJ when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Termination event for ISDN **gc_HoldCall( )** function generated when a Hold Call request is rejected successfully. No action required.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_L2FRAME when using Dialogic® Springware Boards GCEV_L2FRAME event when using Dialogic® DM3 Boards	Notification event (not maskable) generated when an incoming data link layer 2 access message is received. Use **gc_GetFrame( )** function to retrieve the received frame.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_L2NOBFFR when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (not maskable) generated when no free space (buffer) is available for an incoming layer 2 access message. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_NOTIFY when using Dialogic® Springware Boards GCEV_NOTIFY event when using Dialogic® DM3 Boards	Notification event (not maskable) generated when an incoming Notify message is received. Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_NOUSRINFOBUF when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Notification event (not maskable) indicates that the incoming user-to-user information element (UUI) is discarded. An incoming UUI is not accepted until the existing UUI is read by the application. No action required.
GCEV_NSI when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (not maskable) generated when a Network Specific Information (NSI) message is received (DPNSS protocol only). Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_PROCEEDING when using both Dialogic® Springware and DM3 Boards	Notification event (enabled by default) generated when an incoming Proceeding message is received. Use **gc_SetEvtMsk( )** function to clear the mask so that the application is notified when the event occurs.
GCEV_PROGRESSING when using both Dialogic® Springware and DM3 Boards	Notification event (enabled by default) generated when an incoming Progress message is received. Use **gc_SetEvtMsk( )** function to mask event.
GCEV_REQANI when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Termination event for ISDN **gc_ReqANI( )** function generated when ANI information is received from network. (Applies to AT&T ANI-on-demand feature only.) No action required.
GCEV_RESETLINEDEV when using both Dialogic® Springware and DM3 Boards	Termination event for the asynchronous mode **gc_ResetLineDev( )** function. Application must issue a new **gc_WaitCall( )** function to receive the next incoming call on the channel.
GCEV_RESTARTFAIL when using both Dialogic® Springware and DM3 Boards	Termination event for ISDN indicating that the **gc_ResetLineDev( )** function failed. Use the **gc_ResultValue( )** function to retrieve the reason for failure.

Table 8. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_RETRIEVEACK when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Termination event for ISDN **gc_RetrieveCall( )** function generated when a Retrieve Call request is acknowledged successfully. No action required.
GCEV_RETRIEVECALL when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Unsolicited event (not maskable), generated when the call is retrieved successfully from the HOLD state. Use the **gc_RetrieveAck( )** or the **gc_RetrieveRej( )** function to respond.
GCEV_RETRIEVEREJ when using Dialogic® DM3 Boards (all supported ISDN protocols) and Dialogic® Springware Boards (NTT, BRI, DPNSS, and QSIG protocols only)	Termination event for ISDN **gc_RetrieveCall( )** function generated when a Retrieve Call request is rejected successfully. No action required.
GCEV_SETBILLING when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Termination event for ISDN **gc_SetBilling( )**; generated when billing information for the call is acknowledged by the network. (Applies to AT&T ANI-on-demand feature only.) No action required.
GCEV_SETCHANSTATE when using both Dialogic® Springware and DM3 Boards	Termination event for the asynchronous mode **gc_SetChanState( )** function. Unsolicited event (not maskable) generated when the status of the B channel changes or a Maintenance message is received from the network. Use **gc_GetLineDevState( )** to retrieve B channel status. Use **gc_ResultValue( )** and **gc_ResultMsg( )** to retrieve a cause code and a description of the cause.
GCEV_SETUP_ACK when using both Dialogic® Springware and DM3 Boards	Notification event (enabled by default) generated when an incoming setup ACK (acknowledge) message is received. No action required.
GCEV_TRANSFERACK when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (enabled by default) generated when a Transfer Acknowledge message is received from the network (DPNSS protocol only). Indicates that the network accepted a request to transfer a call. No action required.
GCEV_TRANSFERREJ when using Dialogic® Springware Boards **Note**: Not supported when using Dialogic® DM3 Boards.	Unsolicited ISDN event (enabled by default) generated when a Transfer Reject message is received from the network (DPNSS protocol only). Indicates that the network rejected a request to transfer a call. No action required.

Table 8. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_TRANSIT when using both Dialogic® Springware and DM3 Boards	Unsolicited ISDN event (enabled by default) generated when messages are sent via a call transferring party to the destination party after a transfer call connection is completed (DPNSS protocol only). No action required.
GCEV_USRINFO when using both Dialogic® Springware and DM3 Boards	Unsolicited ISDN event (not maskable) generated when an incoming User Information message is received; for example, in response to a **gc_SndMsg( )** function call in which the **msg_type** specified is SndMsg_UsrInformation. Indicates that a User-to-User Information (UUI) event is coming. Use **gc_GetCallInfo( )** function to retrieve call information.

2.7 ISDN-Specific Extension IDs

The Dialogic® Global Call Software provides a common interface to multiple network interface libraries for features that are abstracted across multiple call control libraries. The Feature Transparency and Extension (FTE) module of Global Call provides the flexibility to extend the Global Call API to access all technology or protocol-specific features unique to any given network interface. For further details, refer to the *Dialogic® Global Call API Programming Guide*.

To use one of these supported features directly through the Global Call API, the **gc_Extension**( ) function is called with an extension function identifier, **ext_id**, defined in this section for ISDN. If the extension function is supported and called in asynchronous mode, relevant information is returned via the call control library through the GCEV_EXTENSIONCMPLT termination event. Network event notification is returned via the call control library through the GCEV_EXTENSION event. For more information on the **gc_Extension**( ) function, the GCEV_EXTENSIONCMPLT event, and the GCEV_EXTENSION event, see the *Dialogic® Global Call API Programming Guide*.

Table 9 provides a list of the extension IDs for ISDN and indicates whether the ID is supported in synchronous and/or asynchronous mode, and if there are termination events.

Table 9. ISDN Extension IDs

Extension ID	Mode	Termination Event
GCIS_EXID_CALLPROGRESS when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, call progress can be sent using **gc_SndMsg( )**.	Sync	No
GCIS_EXID_GETBCHANSTATE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, B channel state can be retrieved using **gc_GetLineDevState( )**.	Sync	No

Table 9. ISDN Extension IDs (Continued)

Extension ID	Mode	Termination Event
GCIS_EXID_GETDCHANSTATE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 boards, D channel state can be retrieved using **gc_GetLineDevState( )**.	Sync	No
GCIS_EXID_GETDLINKSTATE when using Dialogic® Springware Boards	Sync	No
GCIS_EXID_GETENDPOINT when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, retrieving CES and SAPI is not supported.	Sync	No
GCIS_EXID_GETFRAME when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, ISDN frames can be retrieved using **gc_GetFrame( )**.	Sync	No
GCIS_EXID_GETNETCRV when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, the CRV can be retrieved using **gc_GetNetCRV( )**.	Sync	No
GCIS_EXID_GETNONCALLMSG when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, retrieving information associated with the global and null CRN is not supported.	Sync	No
GCIS_EXID_PLAYTONE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, playing a user defined tone is not supported.	Sync, Async	Yes
GCIS_EXID_SETDLINKSTATE when using Dialogic® DM3 and Springware Boards **Note:** When using Dialogic® Springware Boards, only Sync mode is supported. When using Dialogic® DM3 Boards, both the Sync and Async modes are supported; the termination event in Async mode is GCEV_EXTENSIONCMPLT.	Sync, Async	No
GCIS_EXID_SNDFRAME when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, sending frames can be achieved using **gc_SndFrame( )**.	Sync	No
GCIS_EXID_SNDMSG when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, sending a non-call control message can be achieved using **gc_SndMsg( )**. A non-call control message is a message that is not related to call setup or tear down (that is, a message that does not change the call state).	Sync	No
GCIS_EXID_SNDNONCALLMSG when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, sending non-call related messages is not supported.	Sync	No

Table 9. ISDN Extension IDs (Continued)

Extension ID	Mode	Termination Event
GCIS_EXID_STOPTONE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, stopping the playing of a tone is not supported.	Sync, Async	Yes
GCIS_EXID_TONEREDEFINE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, redefining call progress tone attributes is not supported.	None	Yes

2.8 GCEV_EXTENSION Events

There are ISDN-specific Dialogic® Global Call API events, which will eventually be mapped to GCEV_EXTENSION. But to maintain backward compatibility, the Global Call application has the option to choose ISDN-specific events or GCEV_EXTENSION. The default is ISDN-specific events. For more information, refer to Section 4.2, "Operations Performed Using RTCM", on page 137.

Note: When using Dialogic® DM3 Boards, the GCEV_EXTENSION event is not supported. Dialogic® DM3 boards use ISDN-specific events only.

If the application needs to use the new generic call model or extension features, **gc_Start( )** should be called as shown below:

```
CCLIB_START_STRUCT cclib_struct;
GC_START_STRUCT gc_start_struct;
GC_PARM_BLK *parmblk = NULL;

gc_util_insert_parm_val( &parmblk,
                         GCIS_SET_GENERIC,
                         GCIS_PARM_EXTENSIONEVENT,
                         sizeof( char ), 1);

gc_util_insert_parm_val( &parmblk,
                         GCIS_SET_GENERIC,
                         GCIS_PARM_GENERICCALLMODEL,
                         sizeof( char ), 1);
gc_start_struct.num_cclibs = 1;
gc_start_struct.cclib_list = &cclib_struct;
gc_start_struct.cclib_list[0].cclib_name =  "GC_ISDN_LIB";
gc_start_struct.cclib_list[0].cclib_data = parmblk;

if ( gc_Start( &gc_start_struct ) != GC_SUCCESS ) {
    exit(1);
}
gc_util_delete_parm_blk(parmblk);
```

The field extevtdatap of the METAEVENT structure points to EXTENSIONEVT_BLK.

```
typedef struct {
    unsigned char    ext_id;
    GC_PARM_BLK      parmblk;
} EXTENSIONEVTBLK;
```

Table 10 describes the different possible extension IDs in the GCEV_EXTENSION event.

Table 10. GCEV_EXTENSION Events

Event	Description
GCIS_EXEV_CONFDROP when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A DROP request has been received; the request was made by sending the SndMsg_Drop message type via the **gc_Extension(GCIS_EXID_SNDMSG)** function. This event has two different meanings that depend upon the type of call: • Two-party call - the event is a request to disconnect the call. The application should respond by issuing a **gc_DropCall( )**. • Conference call - the event is a request to remove the last party that was added to the conference. The application needs to respond to this request with either a SndMsg_DropAck or SndMsg_DropRej message to indicate the acceptance or rejection of the request. If the request is accepted, the party is dropped from the conference. This event only pertains to a Custom BRI 5ESS switch type.
GCIS_EXEV_CONGESTION when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_CONGESTION	A CONGESTION message has been received by the application, indicating that the remote end is not ready to accept incoming user information. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.
GCIS_EXEV_DIVERTED when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	NAM with divert information has been received by the application. An outgoing call has been successfully diverted to another station.
GCIS_EXEV_DROPACK when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 boards, this event is not supported.	The network has honored a DROP request for a conference call; the request was made by sending the SndMsg_Drop message type via the **gc_Extension(GCIS_EXID_SNDMSG)** function. The event is sent on the corresponding line device. This event pertains only to a Custom BRI 5ESS switch type.
GCIS_EXEV_DROPREJ when using Dialogic® Springware boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	The network has not honored a DROP request for a conference call. The event is sent on the corresponding line device.This event pertains only to a Custom BRI 5ESS switch type.
GCIS_EXEV_FACILITY when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_FACILITY	A FACILITY REQUEST message has been received by the application.
GCIS_EXEV_FACILITY_ACK when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A FACILITY_ACKNOWLEDGEMENT message has been received by the application.
GCIS_EXEV_FACILITY_REJ when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A FACILITY_REJECT message has been received by the application.

Table 10. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_FACILITYGLOBAL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_FACILITY message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_FACILITYNULL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_FACILITY message was received containing a Dummy (NULL) CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_INFOGLOBAL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_INFORMATION message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_INFONULL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_INFORMATION message was received containing a NULL CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_L2BFFRFULL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	Reserved for future use.
GCIS_EXEV_L2FRAME when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_L2FRAME	A data link layer frame has been received by the application. The application should use the **gc_Extension(GCIS_EXID_GETFRAME)** function to retrieve the received frame. It is the application's responsibility to analyze the contents of the frame or look into the extension event data.
GCIS_EXEV_L2NOBFFR when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	There are no buffers available to save the incoming frame.
GCIS_EXEV_NOFACILITYBUF when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	Facility buffer is not ready.
GCIS_EXEV_NOTIFY when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_NOTIFY	A NOTIFY message has been received by the application. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.

Table 10. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_NOTIFYGLOBAL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_NOTIFY message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_NOTIFYNULL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An ISDN_NOTIFY message was received containing a Dummy (NULL) CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_NOUSRINFOBUF when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	User IE buffer is not ready.
GCIS_EXEV_NSI when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A Network Specific Indication (NSI) message was received from the network. The application should use **gc_GetCallInfo( )** to retrieve the NSI string(s) or look into the extension event data.
GCIS_EXEV_PLAYTONE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	User-defined tone successfully played.
GCIS_EXEV_PLAYTONEFAIL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	Request to play user-defined tone failed.
GCIS_EXEV_PROGRESSING when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_PROGRESSING	A PROGRESS message has been received by the application. By default, the firmware will send this event to the application. The application may block this event by clearing the CCMSK_PROGRESS bit. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.
GCIS_EXEV_STATUS when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A STATUS message has been received from the network.
GCIS_EXEV_STATUS_ENQUIRY when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A STATUS_ENQ message has been received from the network.

Table 10. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_STOPTONE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	The tone operation was terminated.
GCIS_EXEV_STOPTONEFAIL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	The request to terminate the playing of a tone failed.
GCIS_EXEV_TIMER when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	An unsolicited event indicating that a timer has expired.
GCIS_EXEV_TONEREDEFINE when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	The tone(s) in the firmware tone template table was successfully redefined.
GCIS_EXEV_TONEREDEFINEFAIL when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	The request to redefine tone(s) in the firmware tone template table failed.
GCIS_EXEV_TRANSFERACK when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A TRANSFER ACKNOWLEDGE message was received from the network. The indicated network has accepted a request to transfer a call.
GCIS_EXEV_TRANSFERREJ when using Dialogic® Springware Boards **Note:** When using Dialogic® DM3 Boards, this event is not supported.	A TRANSFER REJECT message was received from the network. The indicated network has rejected a request to transfer a call.

Table 10. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_TRANSIT when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_TRANSIT	After a transfer is established, transit messages are used for relating messages between the originating end and the terminating end.
GCIS_EXEV_USRINFO when using Dialogic® Springware Boards When using Dialogic® DM3 Boards, the equivalent event is GCEV_USRINFO	A USER INFORMATION message has been received by the application, indicating that a user-to-user information (UUI) event is coming. For example, this event is received in response to a **gc_Extension(GCIS_EXID_SNDMSG)** function call, from the far end, in which the msg_type is SndMsg_UsrInformation. Use the **gc_GetCallInfo( )** function to retrieve the UUI or look into the extension event data. Field parmblk of EXTENSIONEVTBLK will hold the following parameters: GCIS_SET_IE, GCIS_PARM_UIEDATA (char array, maximum length can go to MAXLEN_IEDATA): Unprocessed IEs in CCITT format. The IEs are returned as raw data and must be parsed and interpreted by the application.

ISDN Call Scenarios 3

This chapter provides charts describing various call control scenarios, including call setup and tear down, network and application initiated call termination, and requests for various ISDN services, using both asynchronous and synchronous mode programming. The call scenarios are described in the following categories:

3.1 General ISDN Call Scenarios

Generic ISDN call control scenarios include the following:

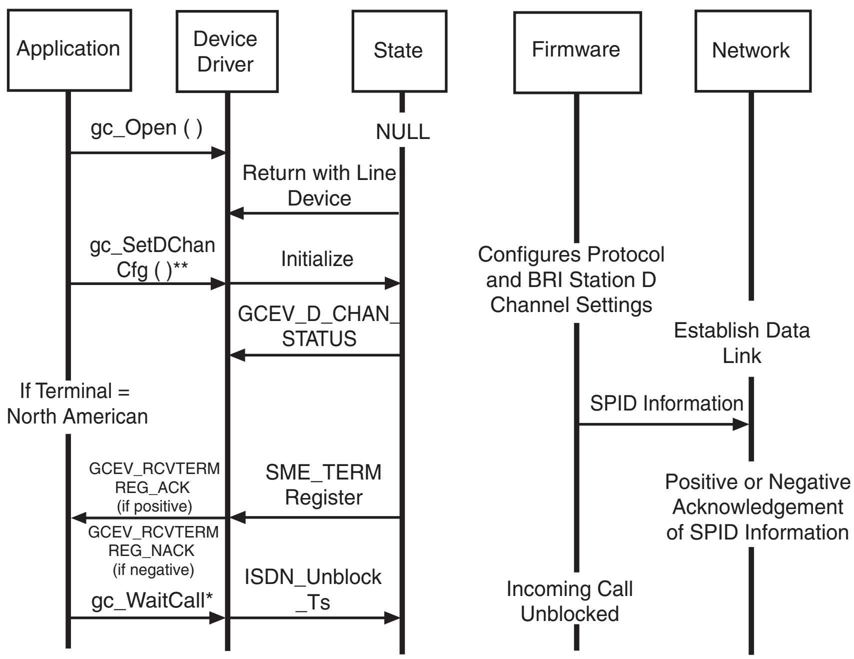
- BRI Channel Initialization and Startup - User Side
- BRI Channel Initialization and Startup - Network Side
- PRI Channel Initialization and Startup
- Network-Initiated Inbound Call (Synchronous Mode)
- Network-Initiated Inbound Call (Asynchronous Mode)
- Network-Terminated Call (Synchronous Mode)
- Network-Terminated Call (Asynchronous Mode)
- Network-Terminated Call When the Application Does Not Drop the Call
- Application-Initiated Outbound Call (Synchronous Mode)
- Application-Initiated Outbound Call (Asynchronous Mode)
- Aborting an Application-Initiated Call
- Application-Terminated Call (Synchronous Mode)
- Application-Terminated Call (Asynchronous Mode)
- Network-Rejected Outbound Call (Asynchronous Mode)
- Application-Rejected Inbound Call (Synchronous Mode)
- Application-Rejected Inbound Call (Asynchronous Mode)
- Glare - Call Collision
- Simultaneous Disconnect from Any State
- Network Facility Request - Vari-A-Bill (Asynchronous Mode)
- Network Facility Request - ANI-on-Demand on an Inbound Call
- Network Facility Request - Advice-of-Charge on Inbound and Outbound Calls
- Application Disconnects Call (Synchronous Mode)

- Network Facility Request - Two B Channel Transfer (Synchronous Mode)
- Non-Call Associated Signaling on Dialogic® Springware Boards (Synchronous Mode)
- Non-Call Associated Signaling on Dialogic® DM3 Boards
- Call Hold and Retrieve Scenarios

3.1.1 BRI Channel Initialization and Startup - User Side

Figure 4 shows the scenario diagram.

Figure 4. BRI Channel Initialization and Startup - User Side

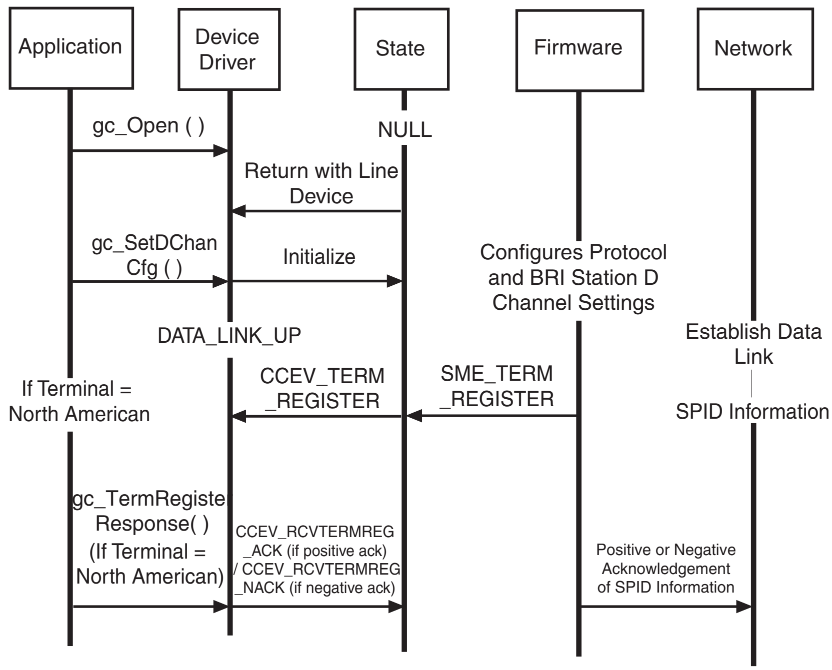


Notes:
* Required for both Synchronous and Asychronous Programming Model.
This process is done once per download.

3.1.2 BRI Channel Initialization and Startup - Network Side

Figure 5 shows the scenario diagram.

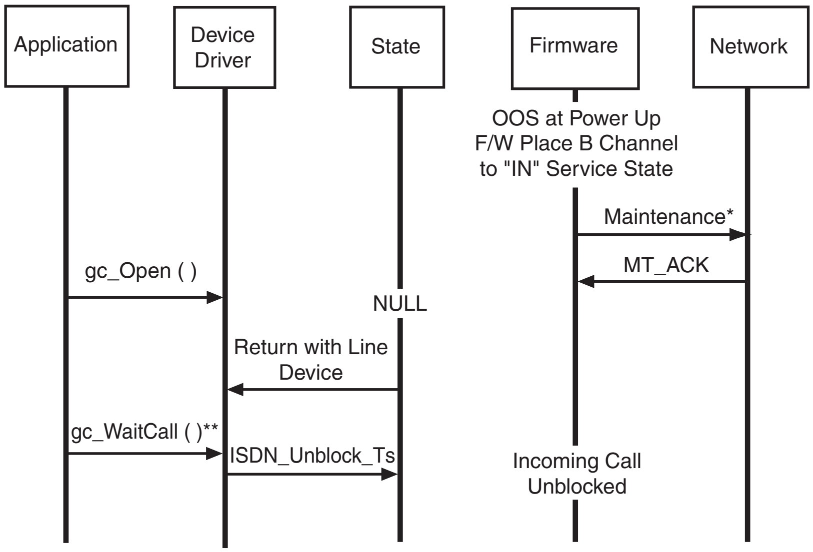
Figure 5. BRI Channel Initialization and Startup - Network Side



3.1.3 PRI Channel Initialization and Startup

Figure 6 shows the scenario diagram.

Figure 6. PRI Channel Initialization and Startup

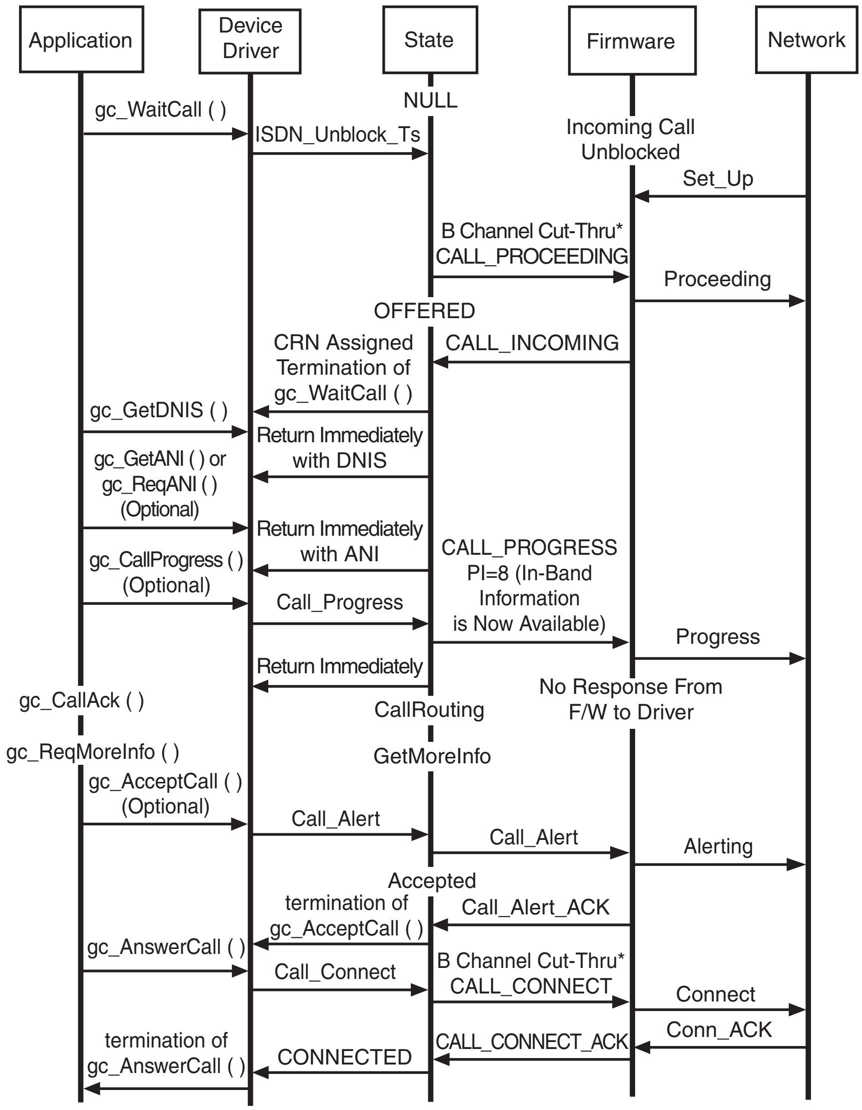


Notes:
* = Optional for TE/NT implementation.
** = Required for both Synchronous and Asychronous Programming Model

3.1.4 Network-Initiated Inbound Call (Synchronous Mode)

Figure 7 shows the scenario diagram.

Figure 7. Network-Initiated Inbound Call (Synchronous Mode)

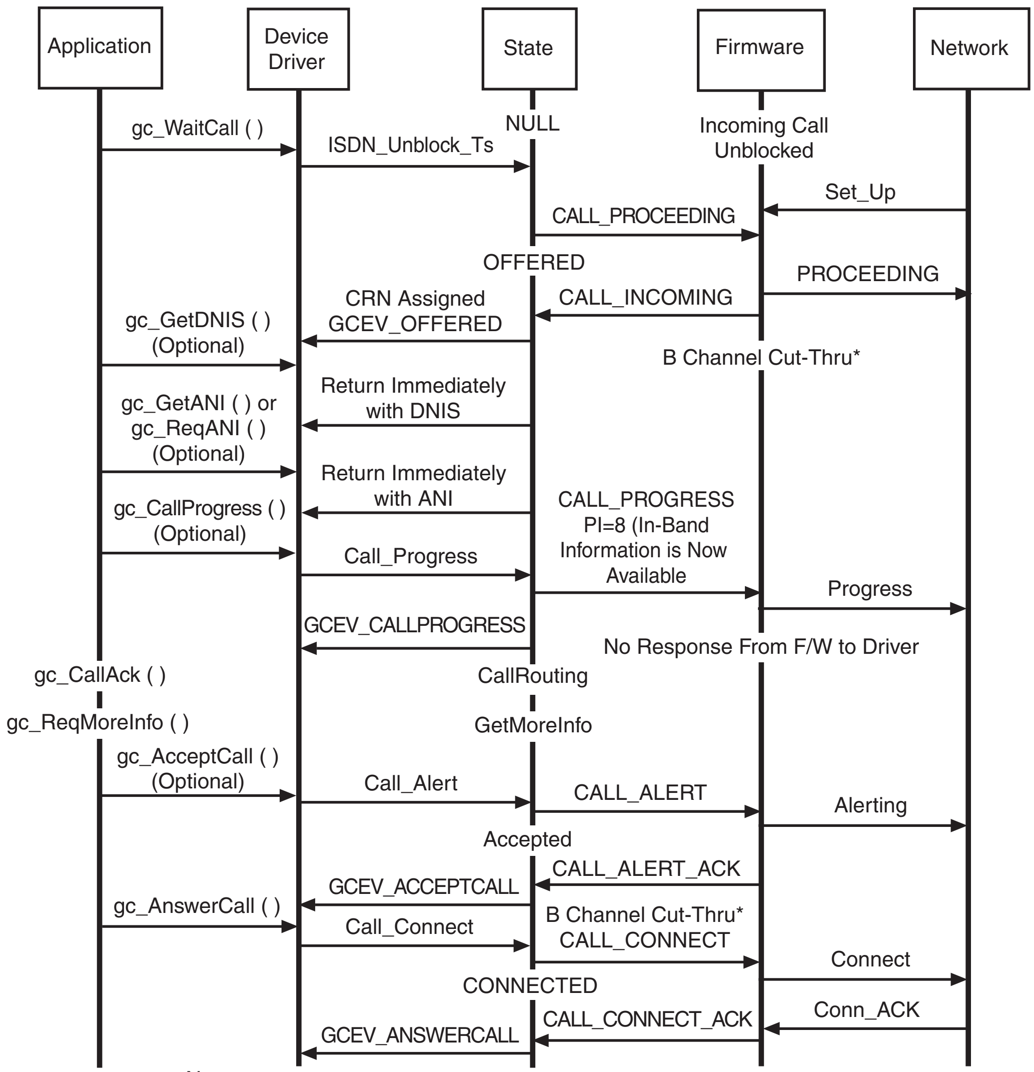


Note: * = Application May Connect a Voice Resource Channel to the B Channel

3.1.5 Network-Initiated Inbound Call (Asynchronous Mode)

Figure 8 shows the scenario diagram.

Figure 8. Network-Initiated Inbound Call (Asynchronous Mode)

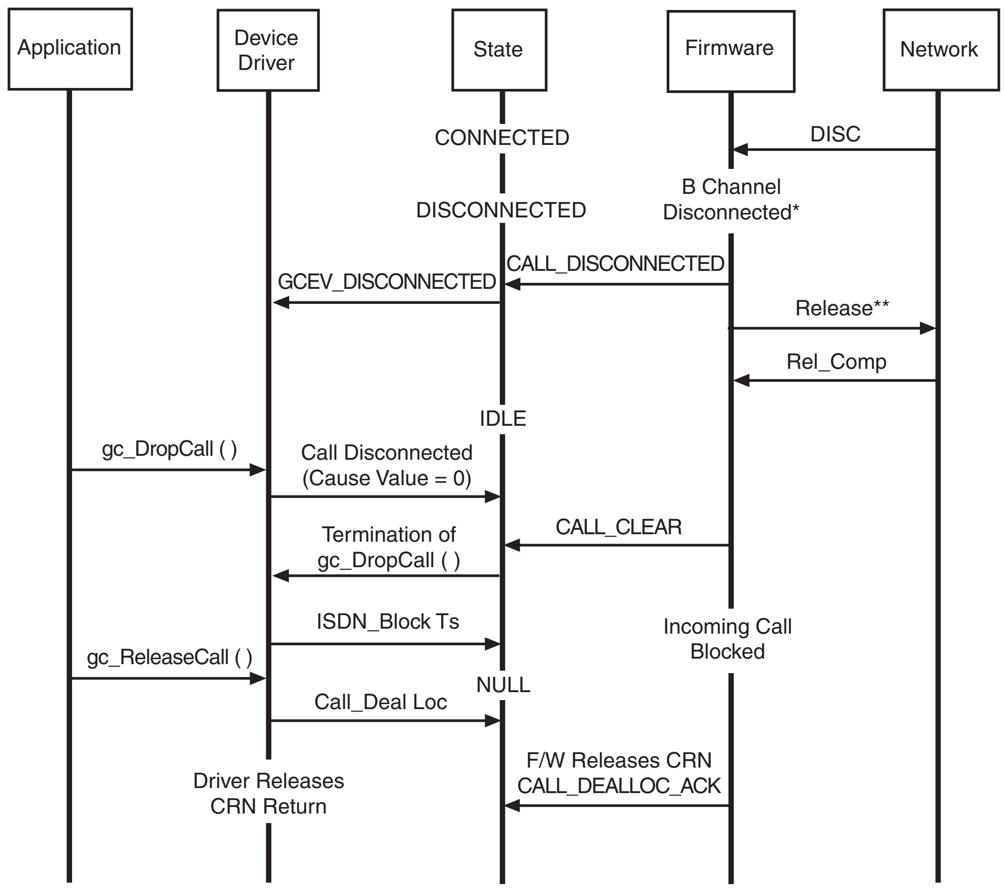


Notes:
* = Application may connect a voice resource channel to the B channel

3.1.6 Network-Terminated Call (Synchronous Mode)

Figure 9 shows the scenario diagram.

Figure 9. Network-Terminated Call (Synchronous Mode)

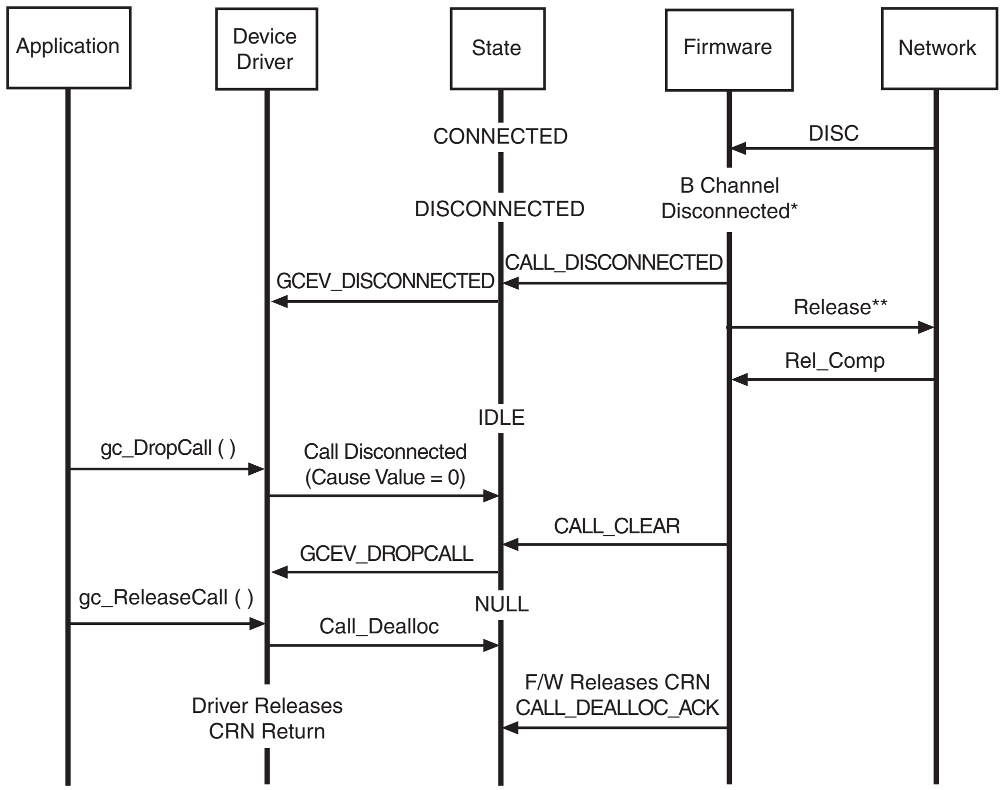


Notes:
* = Firmware Must Ensure That Idle Code is Being Transmitted
** = Drop Call Sent After Release Complete is Received

3.1.7 Network-Terminated Call (Asynchronous Mode)

Figure 10 shows the scenario diagram.

Figure 10. Network-Terminated Call (Asynchronous Mode)

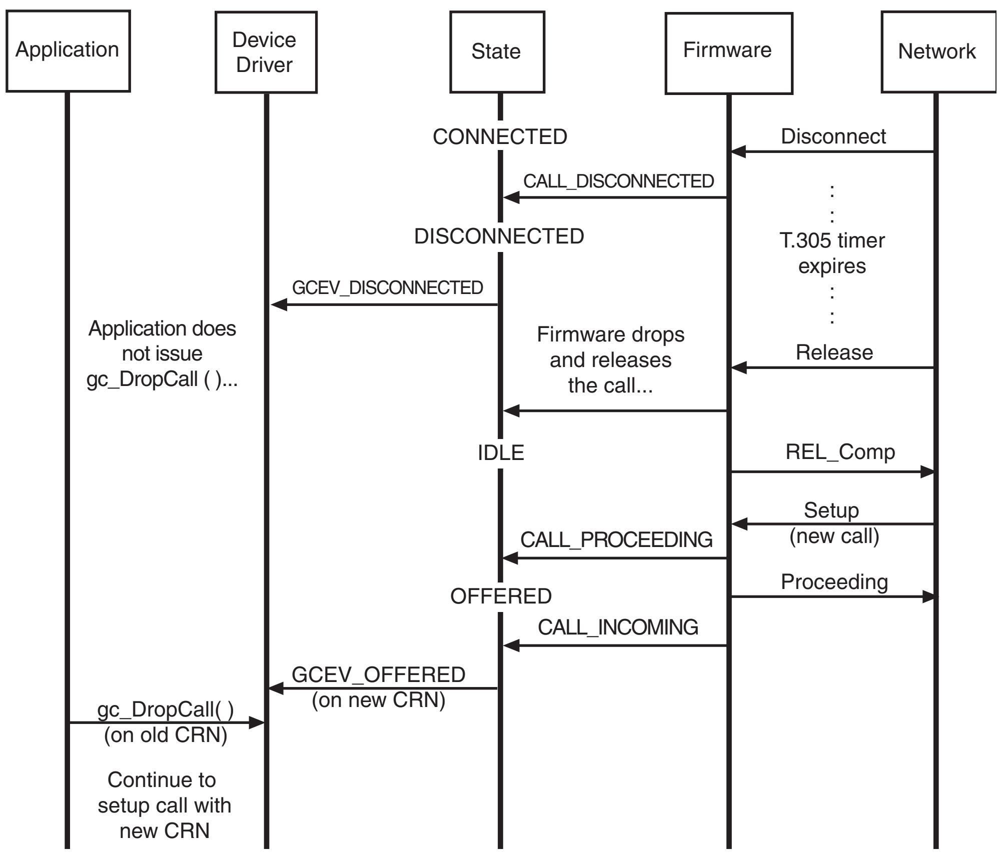


Notes:
* = Firmware Must Ensure That Idle Code is Being Transmitted
** = Drop Call Sent After Release Complete is Received

3.1.8 Network-Terminated Call When the Application Does Not Drop the Call

Figure 11 shows the scenario diagram. In this scenario, the network requests to release the call but the application does not do a drop call before the T.305 timer expires. The network issues a second release request and the firmware automatically drops and releases the call. The CRN for the call is still active, however; and when a new call is received, a second CRN is created so that there are two active CRNs on the line device.

Figure 11. Network-Terminated Call When the Application Does Not Drop the Call

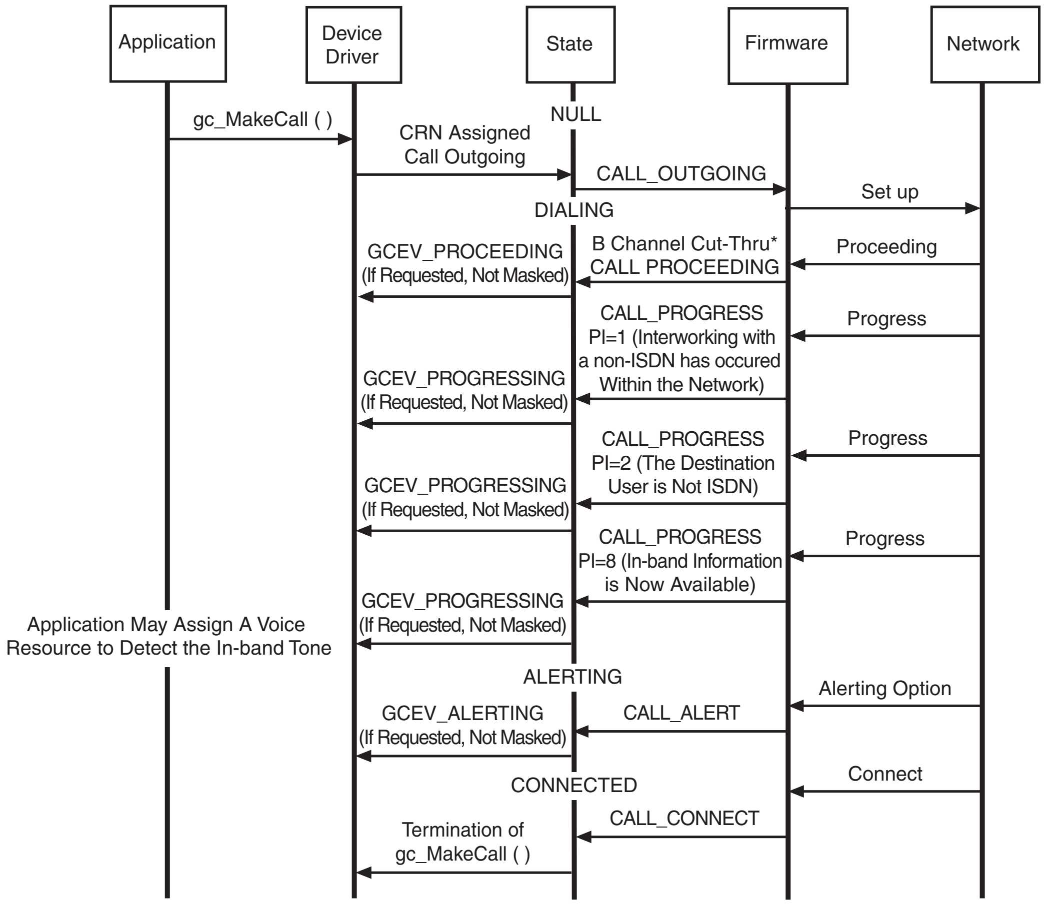


Note: The recommendation in this scenario is to issue a **gc_DropCall( )** on the old CRN and continue processing the call on the new CRN.

3.1.9 Application-Initiated Outbound Call (Synchronous Mode)

Figure 12 shows the scenario diagram.

Figure 12. Application-Initiated Outbound Call (Synchronous Mode)

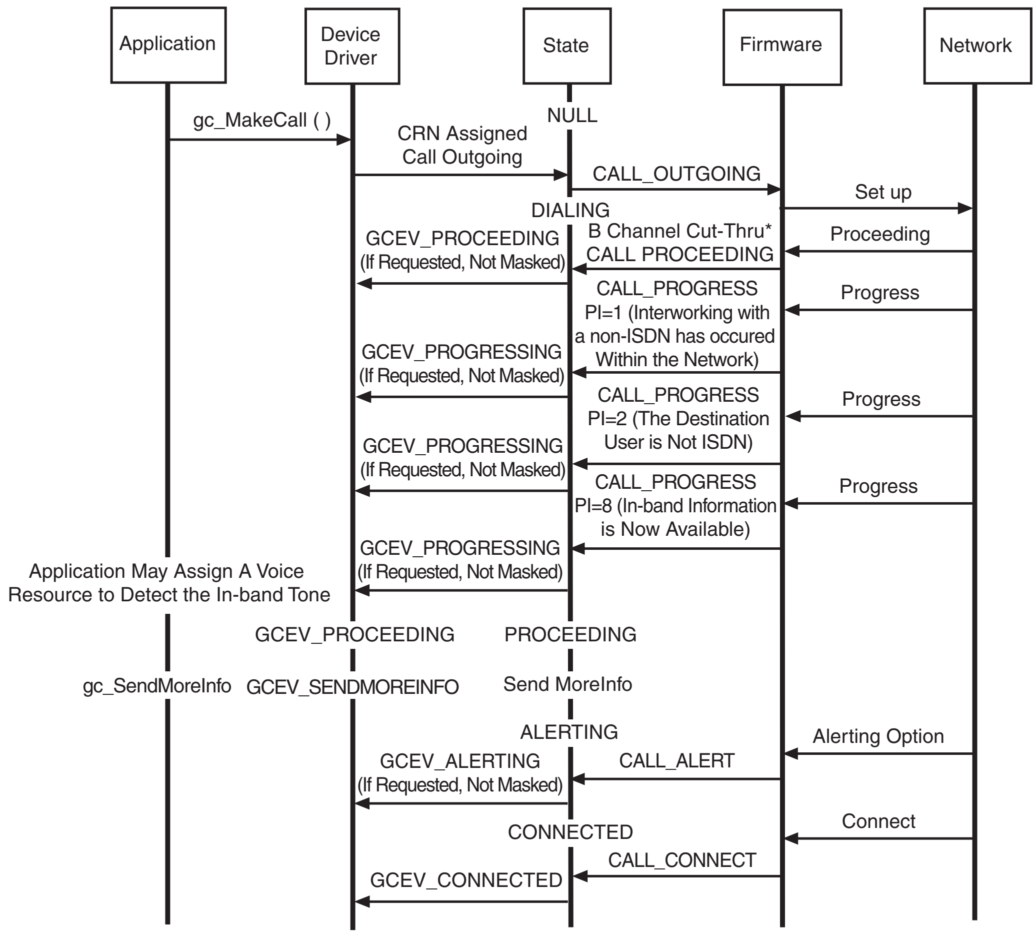


Notes:
* = Application may Connect a Voice Resource Channel to the B Channel

3.1.10 Application-Initiated Outbound Call (Asynchronous Mode)

Figure 13 shows the scenario diagram.

Figure 13. Application-Initiated Outbound Call (Asynchronous Mode)



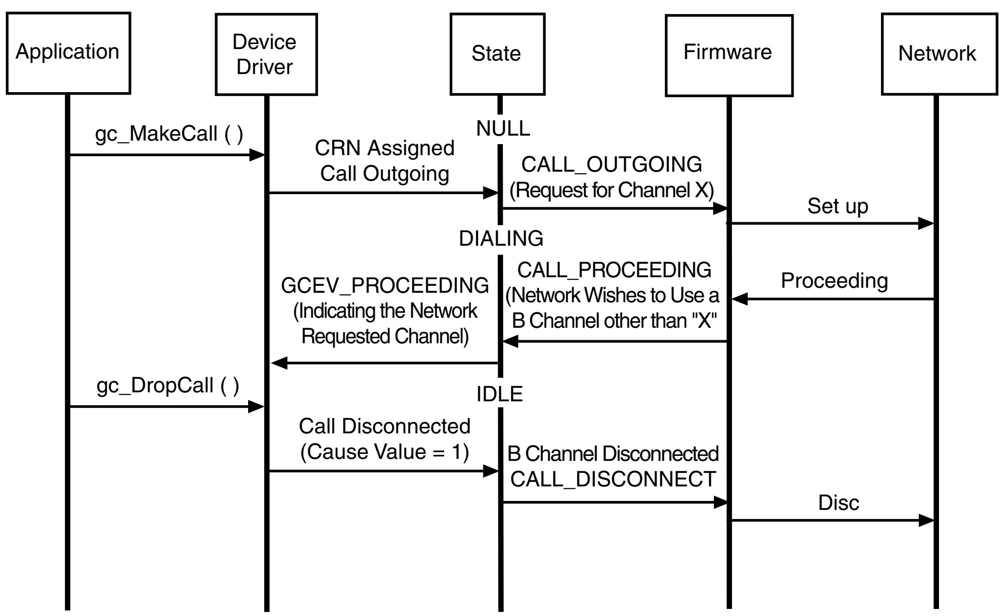
3.1.11 Aborting an Application-Initiated Call

Figure 14 shows the scenario diagram.

Note: B channel negotiation is not currently available.

When B channel negotiation is used in call setup, the application must select the GCEV_PROCEEDING event as the termination point for the **gc_MakeCall( )** function or use the asynchronous programming model. The following scenario illustrates using the asynchronous model to abort the **gc_MakeCall( )** function.

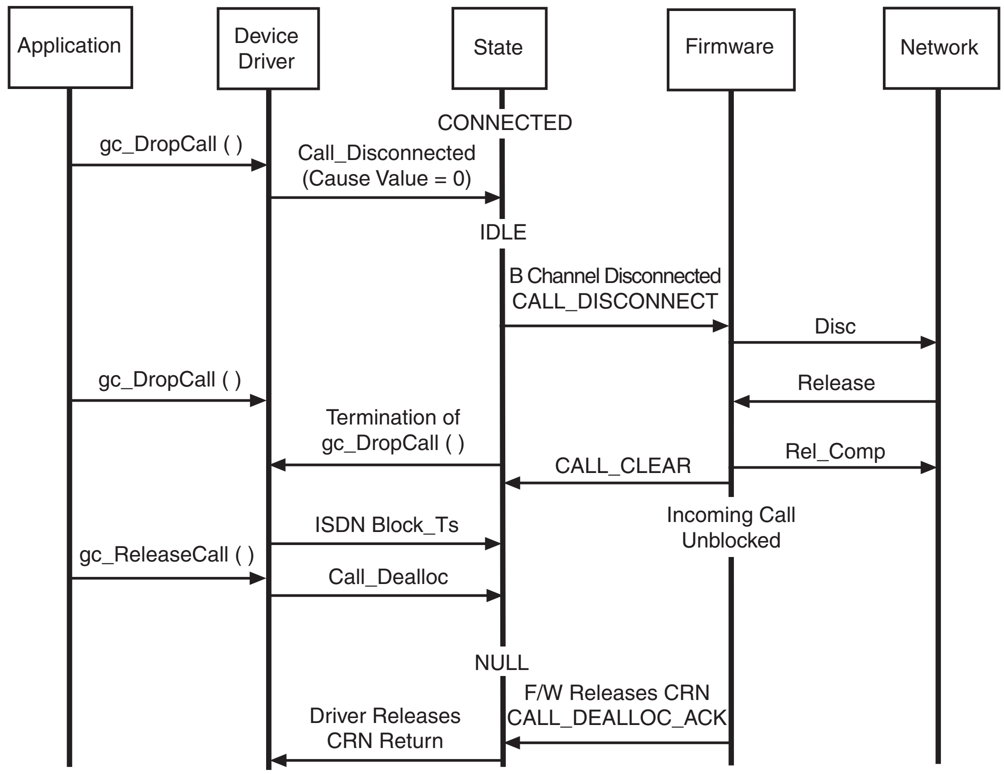
Figure 14. Aborting an Application-Initiated Call



3.1.12 Application-Terminated Call (Synchronous Mode)

Figure 15 shows the scenario diagram.

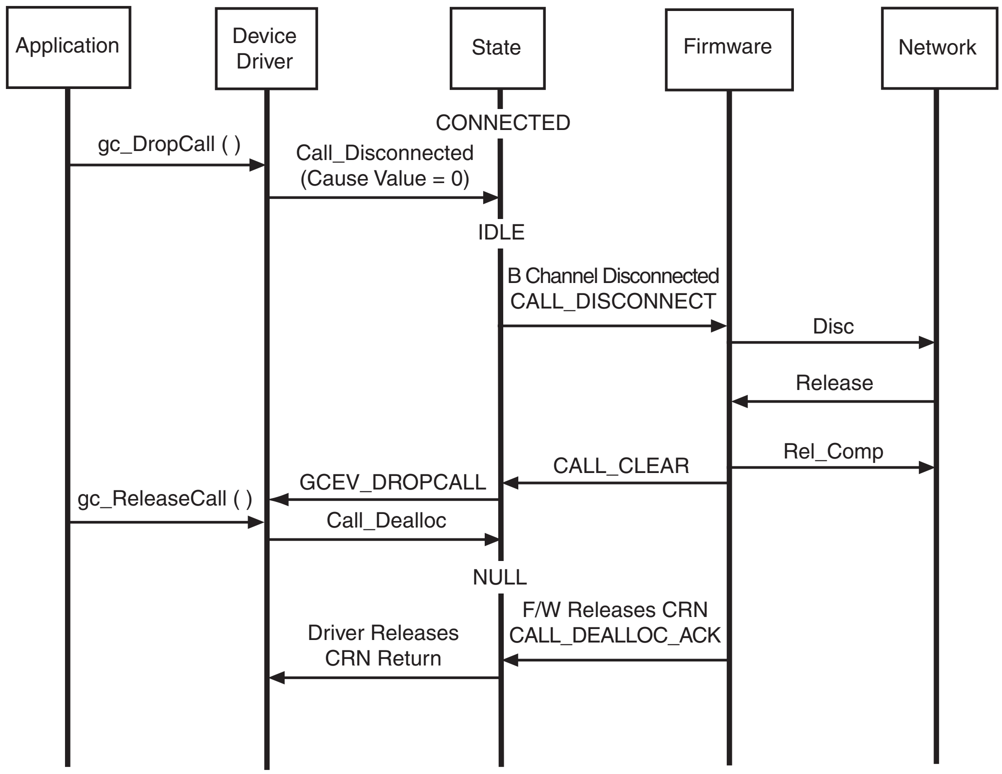
Figure 15. Application-Terminated Call (Synchronous Mode)



3.1.13 Application-Terminated Call (Asynchronous Mode)

Figure 16 shows the scenario diagram.

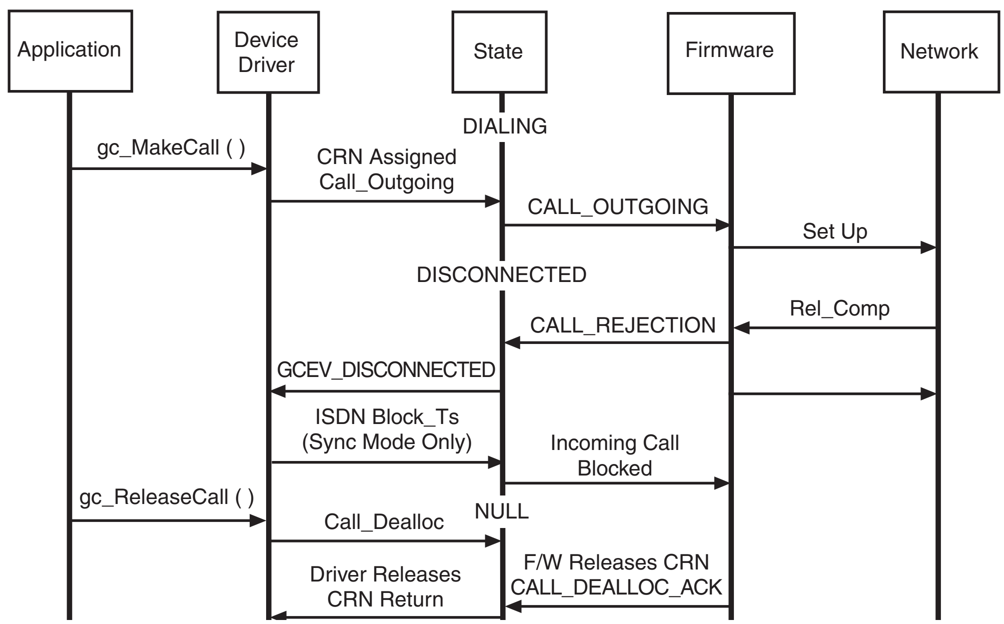
Figure 16. Application-Terminated Call (Asynchronous Mode)



3.1.14 Network-Rejected Outbound Call (Asynchronous Mode)

Figure 17 shows the scenario diagram.

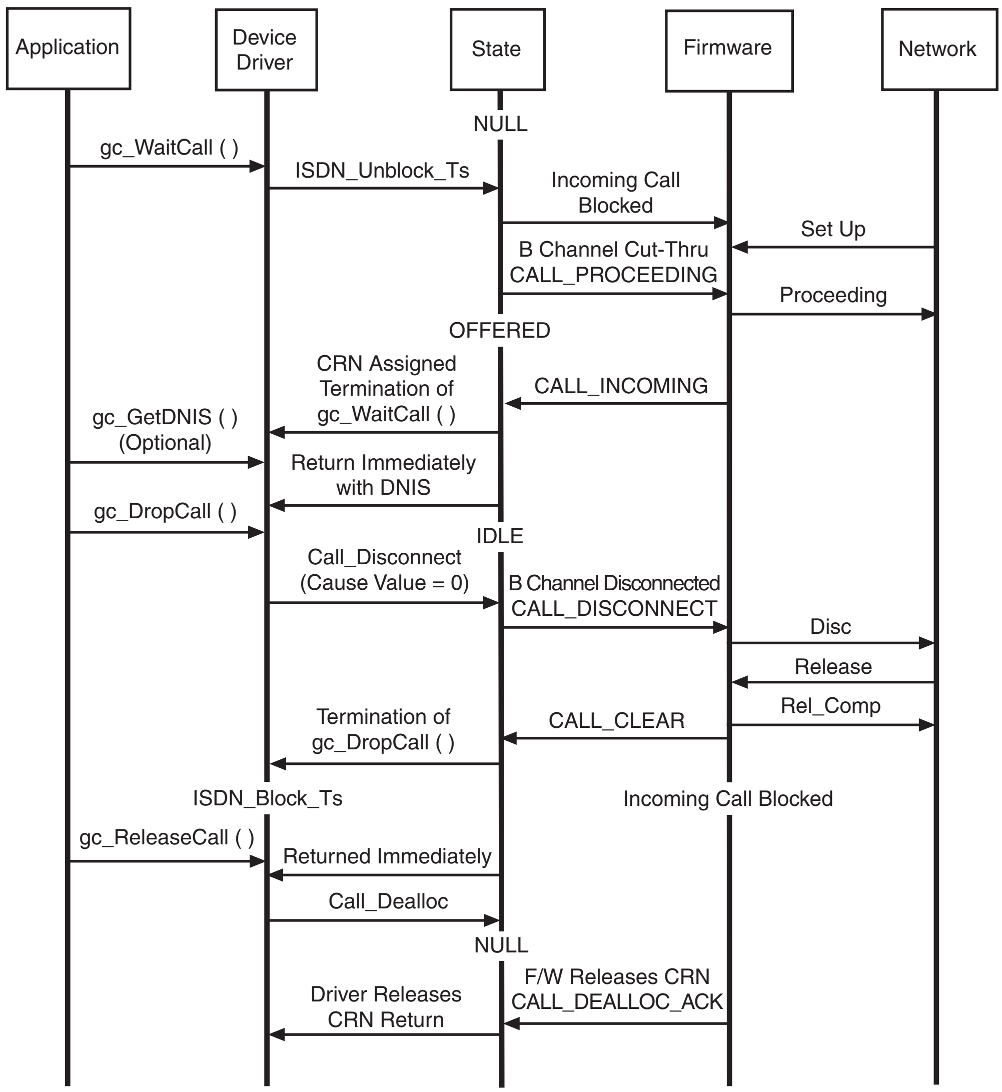
Figure 17. Network-Rejected Outbound Call (Asynchronous Mode)



3.1.15 Application-Rejected Inbound Call (Synchronous Mode)

Figure 18 shows the scenario diagram.

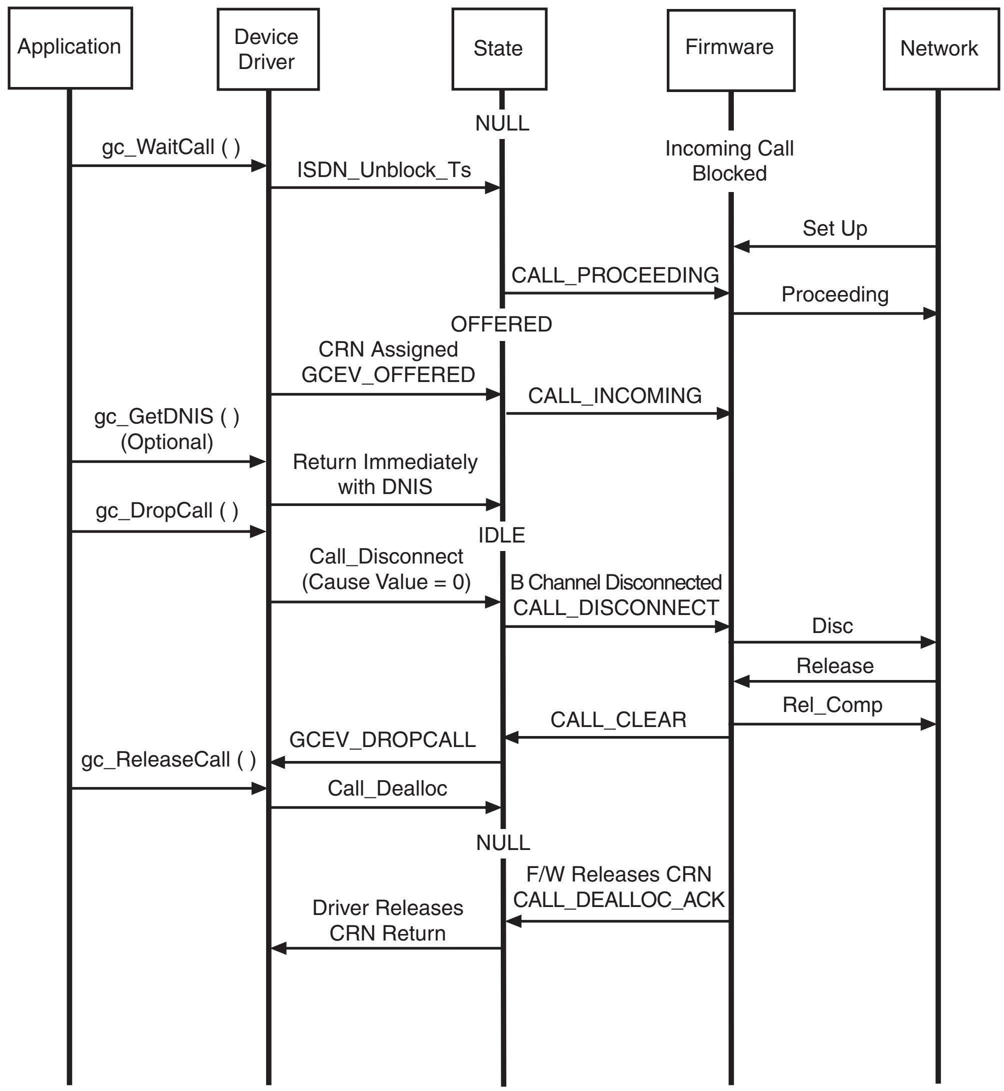
Figure 18. Application-Rejected Inbound Call (Synchronous Mode)



3.1.16 Application-Rejected Inbound Call (Asynchronous Mode)

Figure 19 shows the scenario diagram.

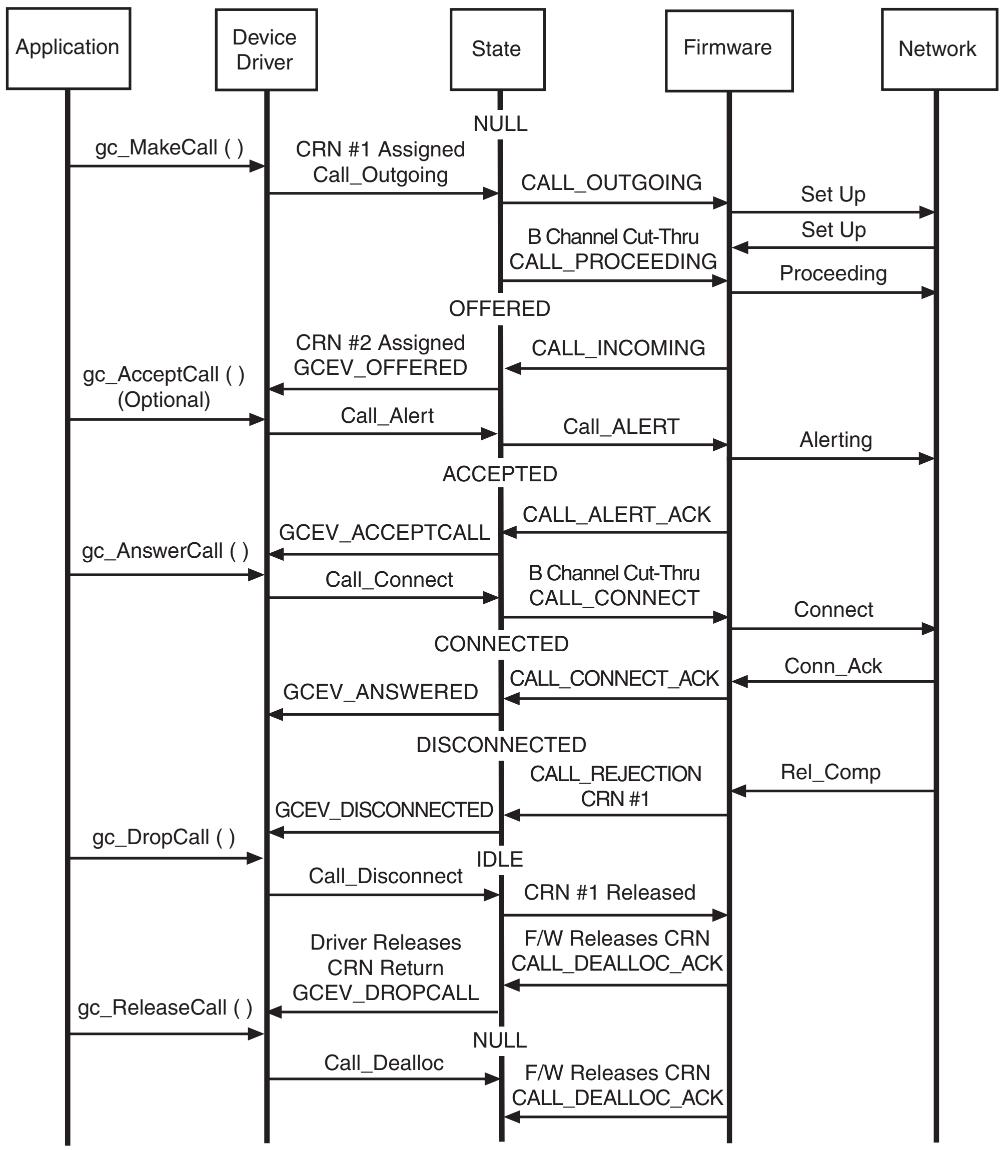
Figure 19. Application-Rejected Inbound Call (Asynchronous Mode)



3.1.17 Glare - Call Collision

A glare condition occurs when both an inbound and outbound call request the same time slot. When glare occurs, the inbound call is assigned the time slot. Figure 20 shows the scenario diagram.

Figure 20. Glare - Call Collision



3.1.18 Simultaneous Disconnect from Any State

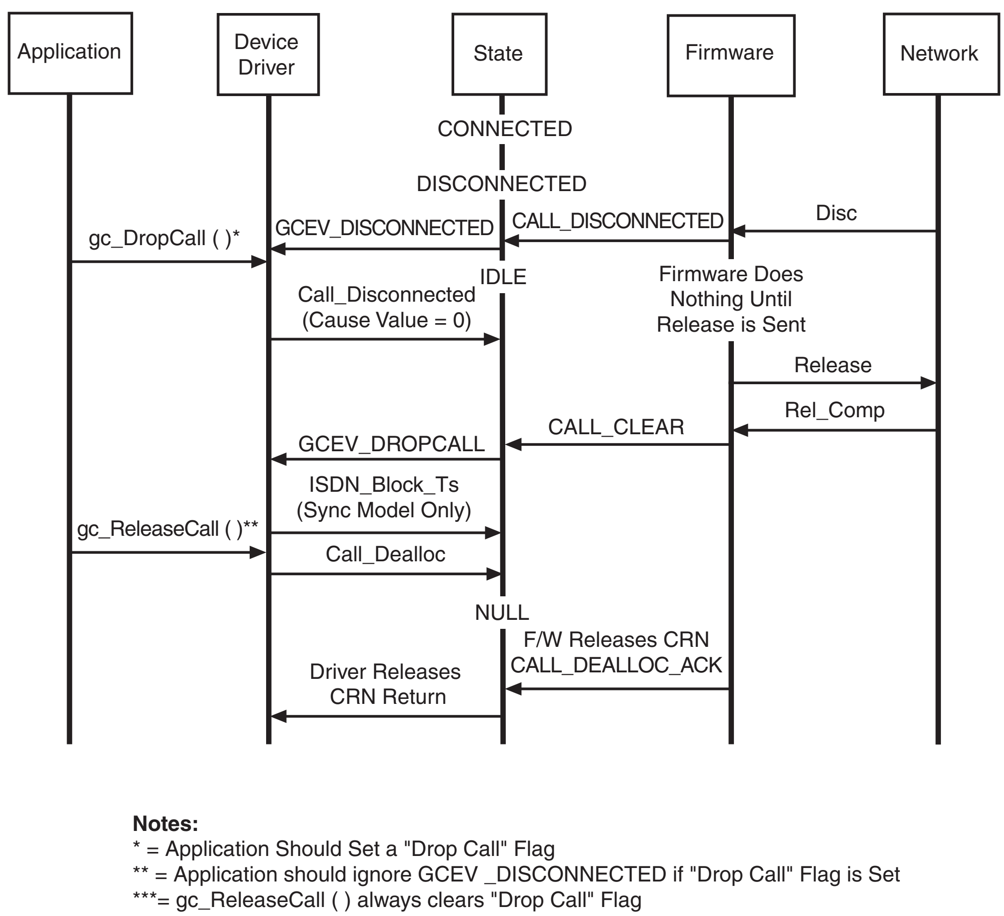
A simultaneous disconnect condition occurs when both the application and the network attempt to disconnect the call. The following scenarios illustrate different disconnect conditions and the asynchronous mode. For synchronous mode, the GCEV_DROPCALL event terminates the **gc_DropCall( )** function.

Scenario 1

Figure 21 shows the scenario diagram, which demonstrates the following:

- Glare at firmware; the firmware detects disconnect condition first, but does nothing until Release command is sent.
- The network disconnects first, while **gc_DropCall( )** function arrives at the firmware **before** a Release command is sent to the network.

Figure 21. Simultaneous Disconnect from Any State Scenario 1

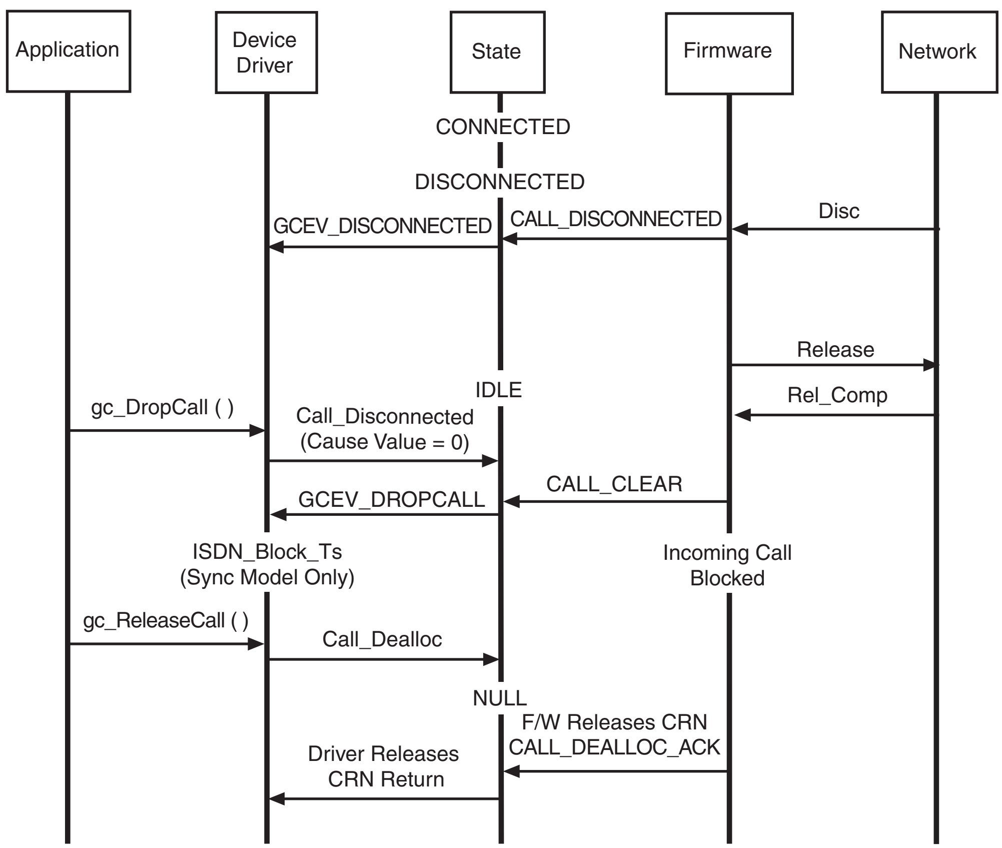


Scenario 2

Figure 22 shows the scenario diagram, which demonstrates the following:

- Glare happens on the line - the firmware receives **gc_DropCall( )** function **after** a Release command is sent to the network.
- The network disconnects first because the **gc_DropCall( )** function arrives at the firmware **after** a Release command is sent to the network.

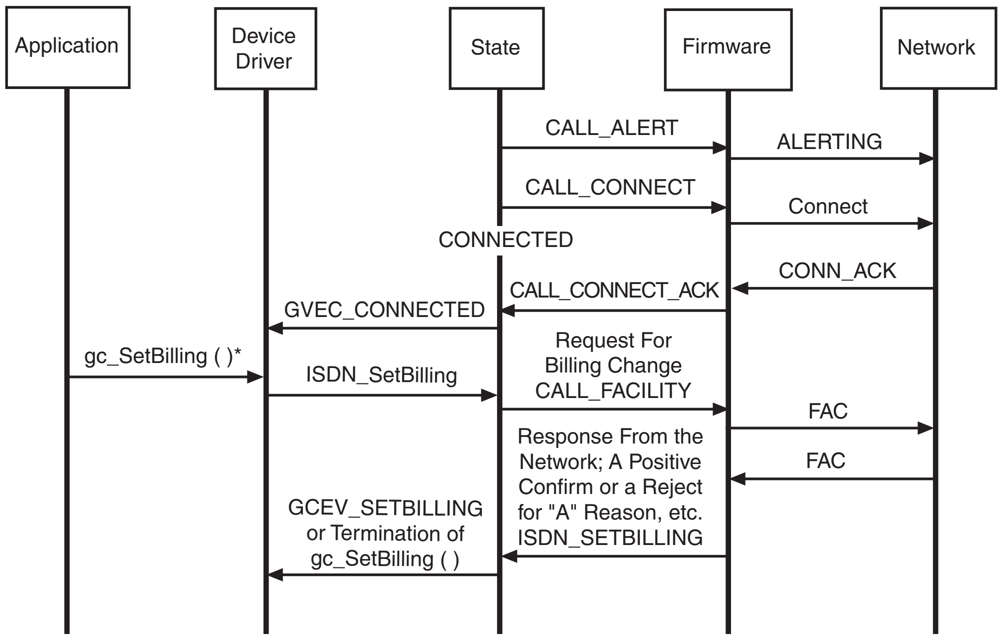
Figure 22. Simultaneous Disconnect from Any State Scenario 2



3.1.19 Network Facility Request - Vari-A-Bill (Asynchronous Mode)

Figure 23 shows the scenario diagram.

Figure 23. Network Facility Request - Vari-A-Bill (Asynchronous Mode)

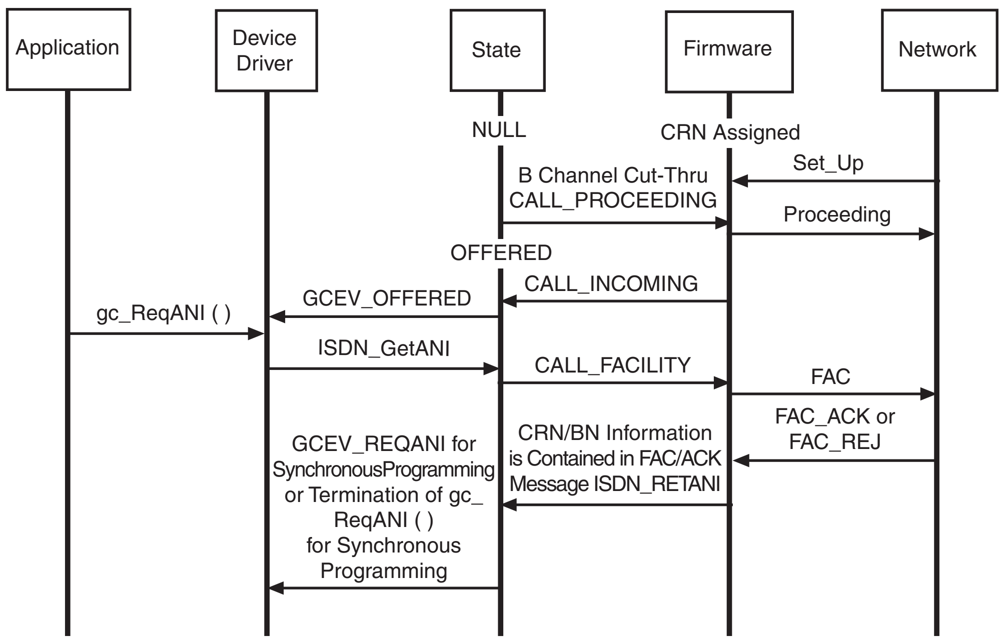


Note: Vari-A-Bill is a Service Option Provided by AT&T

3.1.20 Network Facility Request - ANI-on-Demand on an Inbound Call

The scenario described in this section uses **gc_ReqANI( )** to acquire the caller's ID for either the asynchronous or synchronous mode. It differs from the **gc_GetANI( )** function in the way the function returns. ANI-on-Demand is a service provided by AT&T. Figure 24 shows the scenario diagram.

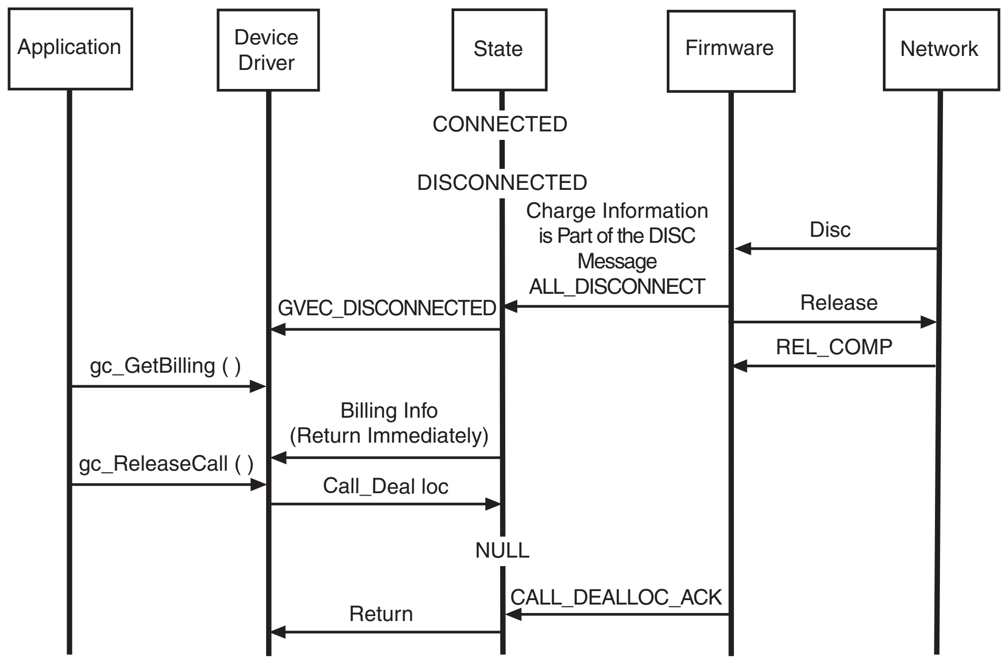
Figure 24. Network Facility Request - ANI-on-Demand on an Inbound Call



3.1.21 Network Facility Request - Advice-of-Charge on Inbound and Outbound Calls

Advice-of-Charge is a service provided by AT&T. Figure 25 shows the scenario diagram.

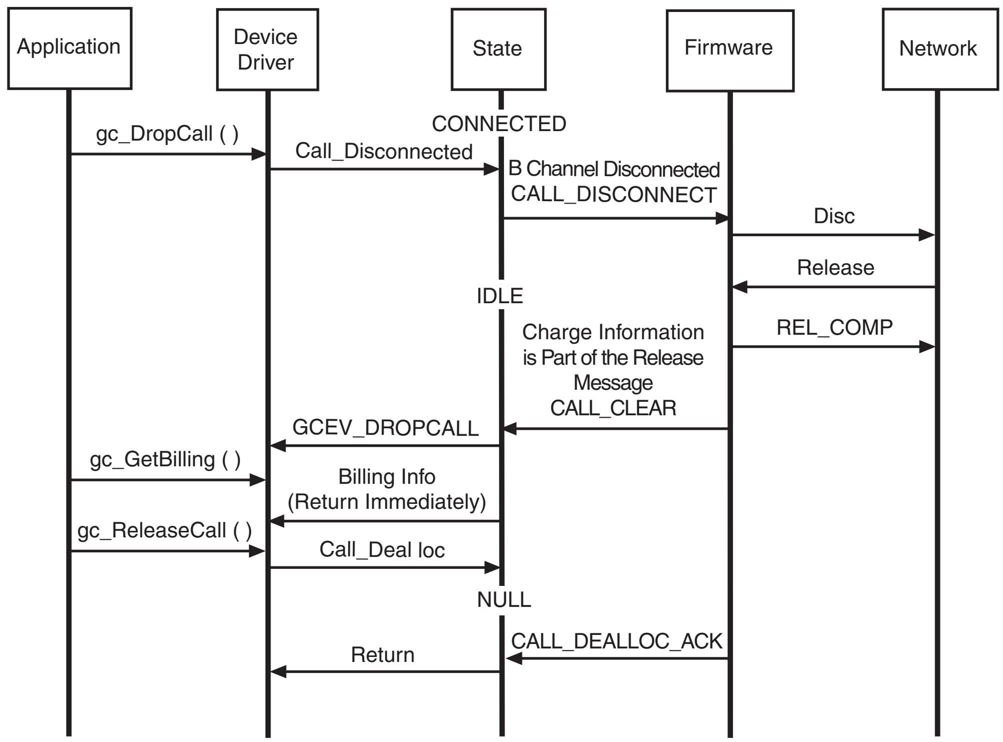
Figure 25. Network Facility Request - Advice-of-Charge on Inbound and Outbound Calls



3.1.22 Application Disconnects Call (Synchronous Mode)

Figure 26 shows the scenario diagram.

Figure 26. Application Disconnects Call (Synchronous Mode)



3.1.23 Network Facility Request - Two B Channel Transfer (Synchronous Mode)

Two B Channel Transfer (TBCT) enables an ISDN PRI user to request the switch to connect together two independent calls on the user's interface. The two calls can be served by the same PRI trunk or by different PRI trunks. If the switch accepts the request, the user is released from the calls and the two calls are connected directly. Billing for the two original calls continues in the same manner as if the transfer had not occurred. As an option, TBCT also allows for transfer notification to the transferred users.

TBCT works only when all of the following conditions are met:

- The user subscribes to TBCT (this feature is supported for 5ESS and DMS switches implementing the NI2 protocols only).
- The two calls are of compatible bearer capabilities.
- At least one of the two calls is answered. If the other call is outgoing from the user, it may be either answered or alerting; if the other call is incoming to the user, it must be answered.

To invoke the TBCT feature, send a FACILITY message to the Network containing, among other things, the Call Reference Values (CRVs) of the two calls to be transferred. The **gc_GetNetCRV( )** function allows applications to query the Dialogic® firmware directly for the Network Call Reference Value. (See the *Dialogic® Global Call API Library Reference* for detailed information about using this function.)

When a transferred call is disconnected, the network informs the TBCT controller by sending a NOTIFY message with the Network Call Reference Value. The application receives the GCEV_EXTENSION event (with ext_id = GCIS_EXEV_NOTIFY) event.

Figure 27 and Figure 28 provide scenario diagrams that illustrate the operation of the TBCT feature. The code example that follows Figure 31 uses the **gc_GetNetCRV( )** function to acquire the Call Reference Values (CRVs) of the two calls to be transferred.

Figure 27. TBCT Invocation with Notification and Both Calls Answered

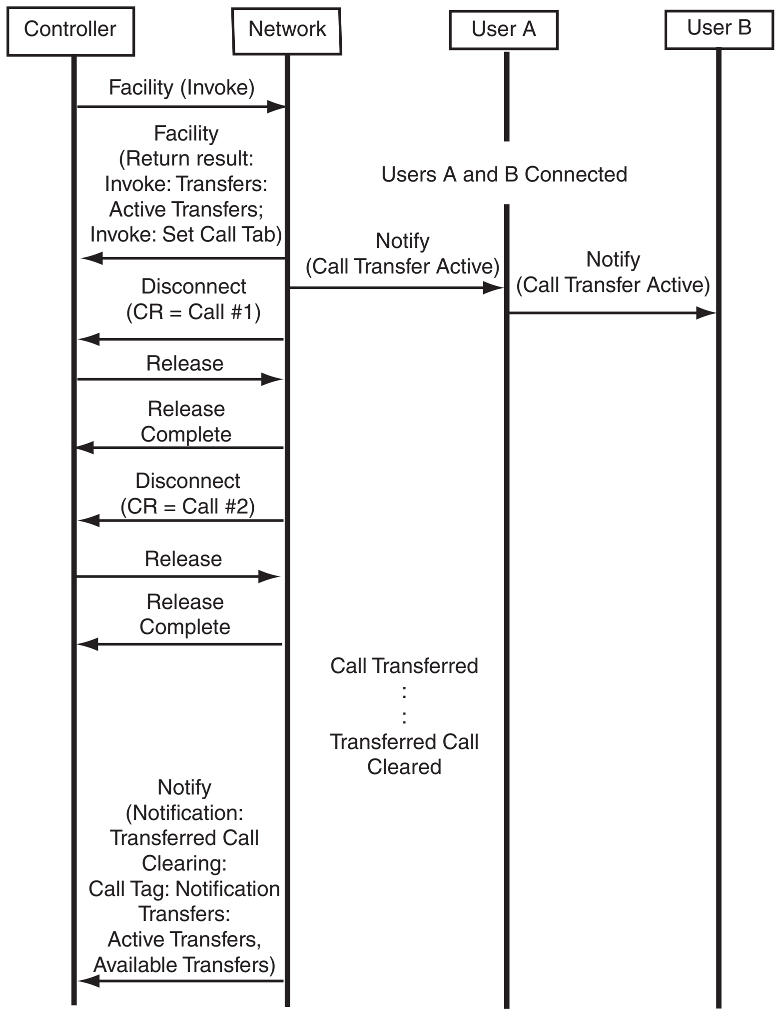


Figure 28. TBCT Invocation with Notification and Call 1 Answered/Call 2 Alerting

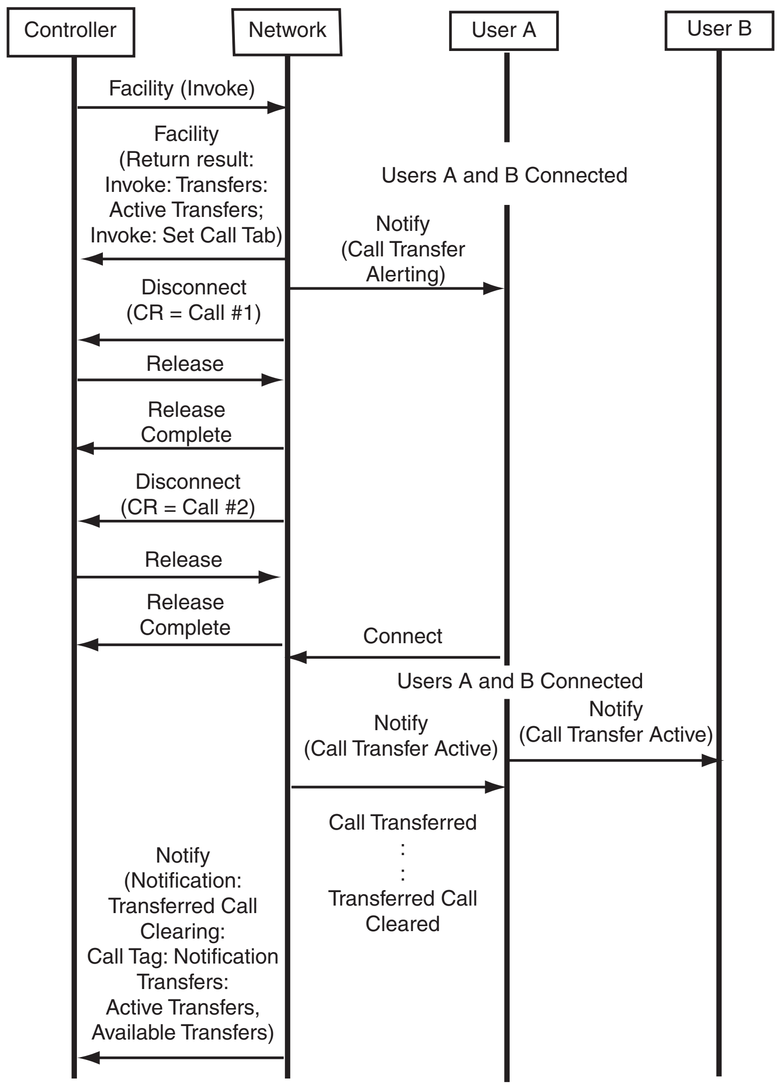


Figure 29, Figure 30, and Figure 31 illustrate the procedures for initiating a TBCT. The scenario is followed by code samples that demonstrate the use of Dialogic® APIs in initiating a TBCT.

Figure 29. Initiating TBCT (Synchronous Mode)

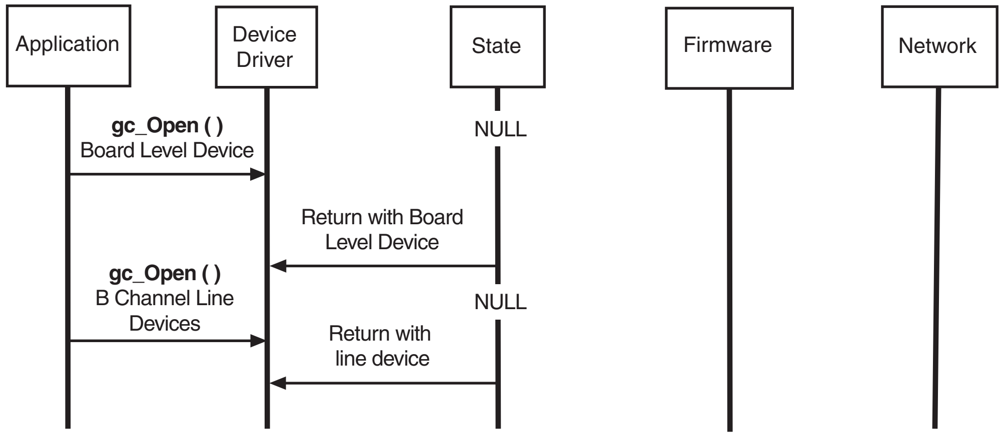


Figure 30. Initiating TBCT with Users A and B Connected (Synchronous Mode)

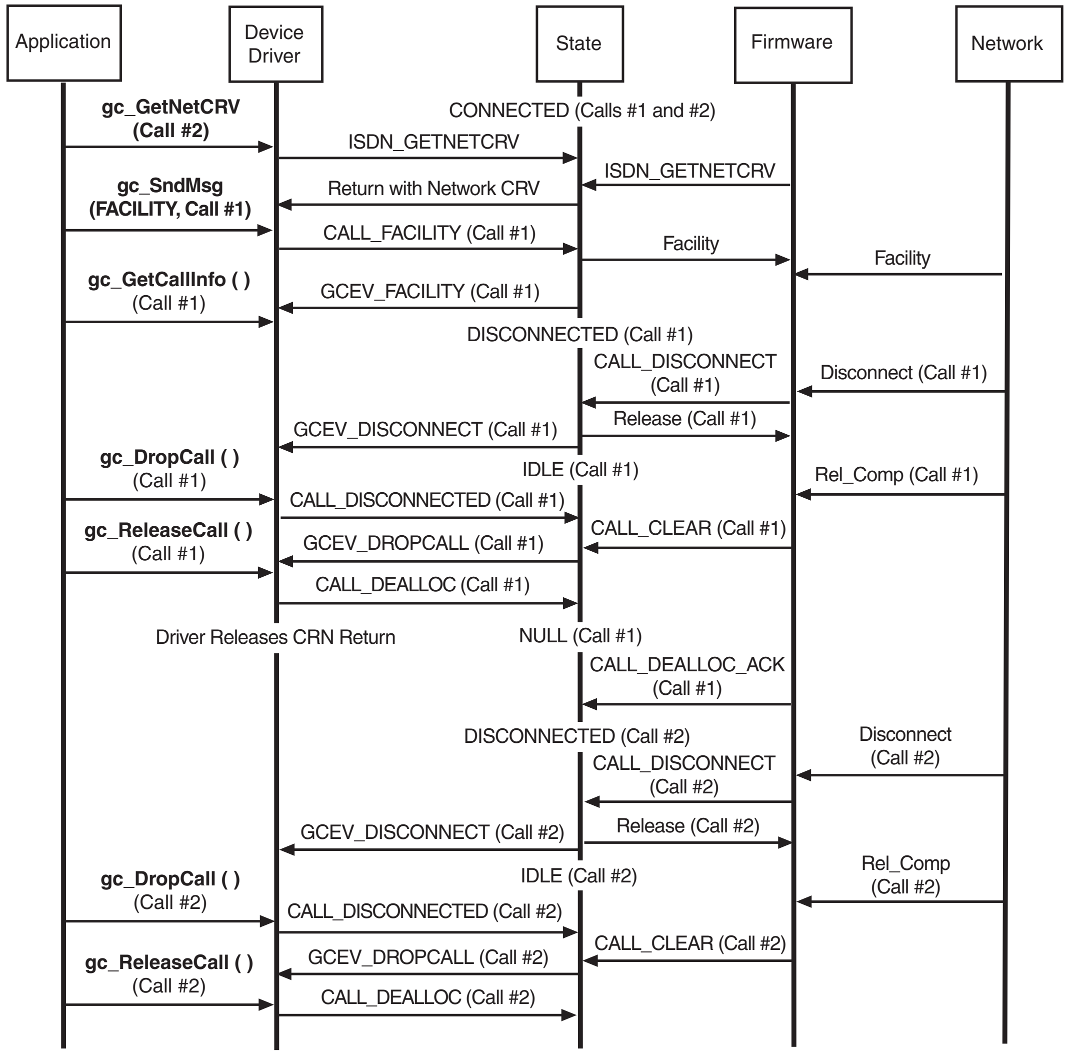
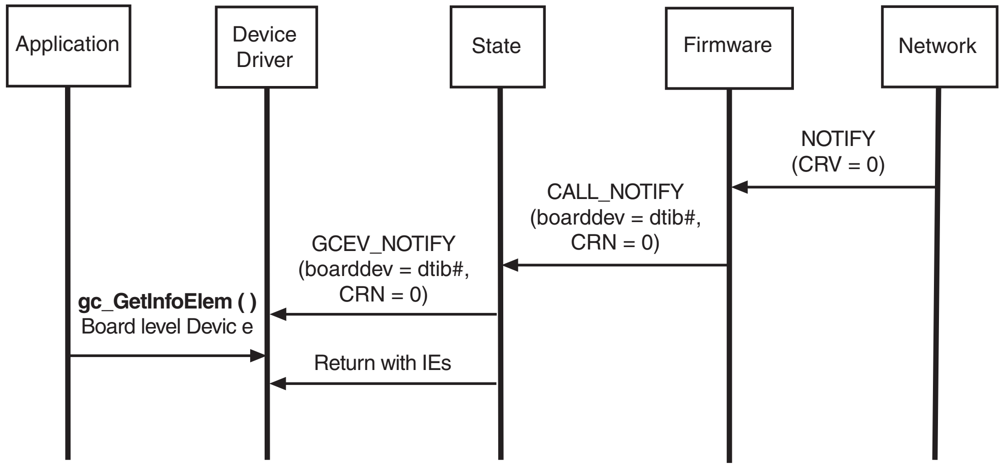


Figure 31. Initiating TBCT with Users A and B Disconnected (Synchronous Mode)



Code Example

The following example code illustrates the use of the Dialogic® Global Call API at various stages of the TBCT call scenario. This code applies to Dialogic® Springware Boards.

1. Opening a board-level device:

```
LINEDEV     dti_dev1_hdl;
    .
    .
rc = gc_Open( &dti_bd_hdl, ":N_dtiB1:P_isdn", 0);
    .
    .
```

2. Retrieving the Network's Call Reference Value:

```
CRN  crn1=0;
unsigned short  crn1_crv=0;
    .
    .
rc = gc_GetNetCRV ( crn1, &crn1_crv );
    .
    .
```

3. Building and sending the Facility message to initiate the TBCT:

```
typedef union {
    struct {
      unsigned char ie_id;                      // Byte 1
      unsigned char length;                     // Byte 2
      unsigned char prot_profile      :5;       // Byte 3, Intel Layout
      unsigned char spare             :2;
      unsigned char extension_1       :1;
      unsigned char comp_type;                  // Byte 4
      unsigned char comp_length;                // Byte 5
      unsigned char comp_data[249];             // Bytes 6 to 254
      .
      .
// Preparing the Facility IE Element
tbct_ie.bits.ie_id          = 0x1C;
tbct_ie.bits.length         = 21;
```

```
tbct_ie.bits.extension_1    = 1;
tbct_ie.bits.spare          = 0x00;
tbct_ie.bits.prot_profile   = 0x11; // Supplementary Service (ROSE)

tbct_ie.bits.comp_type      = 0xA1; // Invoke
tbct_ie.bits.comp_length    = 18;   // Component Length (Data Only)

tbct_ie.bits.comp_data[0]   = 0x02; // Invoke Identifier, tag
tbct_ie.bits.comp_data[1]   = 0x01; // Invoke Identifier, length
tbct_ie.bits.comp_data[2]   = 0x2E; // Invoke Identifier, invoke ie (varies)

tbct_ie.bits.comp_data[3]   = 0x06; // Operation Object, tag
tbct_ie.bits.comp_data[4]   = 0x07; // Operation Object, length
tbct_ie.bits.comp_data[5]   = 0x2A; // Operation Object, Operation Value
tbct_ie.bits.comp_data[6]   = 0x86; // Operation Object, Operation Value
tbct_ie.bits.comp_data[7]   = 0x48; // Operation Object, Operation Value
tbct_ie.bits.comp_data[8]   = 0xCE; // Operation Object, Operation Value
tbct_ie.bits.comp_data[9]   = 0x15; // Operation Object, Operation Value
tbct_ie.bits.comp_data[10]  = 0x00; // Operation Object, Operation Value
tbct_ie.bits.comp_data[11]  = 0x08; // Operation Object, Operation Value

tbct_ie.bits.comp_data[12]  = 0x30; // Sequence, tag
tbct_ie.bits.comp_data[13]  = 0x04; // Sequence, length (varies, combined length
                                    //   of Link & D Channel ID )

tbct_ie.bits.comp_data[14]  = 0x02; // Link ID, tag
tbct_ie.bits.comp_data[15]  = 0x02; // Link ID, length (varies)
tbct_ie.bits.comp_data[16]  = (unsigned char) ((crn2_crv>>8)&0xFF);
// Link ID, linkid value (varies)
tbct_ie.bits.comp_data[17]  = (unsigned char) (crn2_crv&0xFF);
// Link ID, inkid value (varies)

// The D Channel Identifier is Optional
//  tbct_ie.bits.comp_data[18]  = 0x04; // D Channel ID, tag
//  tbct_ie.bits.comp_data[19]  = 0x04; // D Channel ID, length
//  tbct_ie.bits.comp_data[20]  = 0x00; // D Channel ID, dchid (varies)
//  tbct_ie.bits.comp_data[21]  = 0x00; // D Channel ID, dchid (varies)
//  tbct_ie.bits.comp_data[22]  = 0x00; // D Channel ID, dchid (varies)
//  tbct_ie.bits.comp_data[23]  = 0x00; // D Channel ID, dchid (varies)
/*
** Load all the IE's into a single IE block
**   !!NOTE!! - IE must be added in IE ID order!
*/
ie_blk.length = (5 + 18);
for ( ctr = 0; ctr < ie_blk.length; ctr++ ) {
      ie_blk.data[ctr] = tbct_ie.bytes[ctr];
} /* end if */
/*
** Send out a facility message that will execute the transfer
*/
rc = gc_SndMsg( crn2, SndMsg_Facility, &ie_blk );
```

4. Processing the Network response to a TBCT request:

```
typedef union {
    struct {
      unsigned char ie_id;                    // Byte 1
      unsigned char length;                   // Byte 2
      unsigned char prot_profile      :5;     // Byte 3, Intel Layout
      unsigned char spare             :2;
      unsigned char extension_1       :1;
      unsigned char comp_type;                // Byte 4
      unsigned char comp_length;              // Byte 5
      unsigned char comp_data[249];           // Bytes 6 to 254
    } bits;
unsigned char bytes[254];
```

```
} FACILITY_IE_LAYOUT;
      .
FACILITY_IE_LAYOUT  *tbct_ie;
      .
IE_BLK ie_list;
      .
ext_id = (EXTENSIONEVTBLK*) ((metaevt.extevtdatap))->ext_id;
/*assumes 'metaevt' is filled by gc_GetMetaEvent */
switch ( event )
       {
               .
               .
            case GCEV_EXTENSION:
                  switch (ext_id)
                  {
                           .
                           .
                           .
                       case GCIS_EXEV_FACILITY:
                            gc_GetCallInfo( crn2, U_IES, &ie_list);;
                              .
                              .
                              .
                  // retrieve facility IE
            for (ie_len = 2; ie_len < ie_list.length;)
            {
                  if (ie_list[ie_len] == FACILITY_IE)
      // found the facility IE
      {

            tbct_ie = &ie_list[ie_len];   // process the Facility IE
tbct_ie_len = tbct_id->length;
#define FACILITY_IE     0x1C
#define RETURN_RESULT   0xA2
#define RETURN_ERROR    0xA3
#define REJECT          0xA4
#define INVOKE_IDEN_TAG 0x02
                              if (tbct_ie->bits.comp_type == RETURN_RESULT)
                              // network accepted TBCT request{
                              .
                              .
// if subscribed to Notification to Controller, check for Invoke component //
if (tbct_ie->bits.comp_data[0] == INVOKE_IDEN_TAG)
{
      invoke_iden = tbct_ie->bits.comp_data[2];
                            // get invoke identifier
}}
                             else if (tbct_ie->bits.comp_type == RETURN_RESULT)
                                  // network accepted TBCT request
}
                  else
                  {
                  /* if it is not facility IE, go to the next IE */
                  /* if this is single byte IE */
                  if (ie_list[ie_len] & 0x80)
                        /* increment by one byte */
                        ie_len = ie_len + 1;
                  else/* otherwise increment by length of the IE */
                        ie_len = ie_len + ie_list[ie_len + 1];
            }
            }
            break;
                 .
                 .
                 .
                 .
```

5. Processing the Network notification for disconnecting transferred calls:

```
ext_id = (EXTENSIONEVTBLK*) ((metaevt.extevtdatap))->ext_id;
/*assumes 'metaevt' is filled by gc_GetMetaEvent */

switch ( event )
       {
               .
               .
            case GCEV_EXTENSION:
                  switch (ext_id)
                  {
                           .
                           .
                           .
                       case GCIS_EXEV_NOTIFY:
                          gc_GetInfoElem( boarddev, &ie_list );
                              .
                              .
                              .

                          // retrieve Notification IE
                          for (ie_len = 2; ie_len < ie_list.length;)
                          {
                              if (ie_list[ie_len] == NOTIFICATION_IE)
                              // found the Notification IE
                              {

                              }
                              else
                              {
                                  /* if it is not facility IE, go to the next IE */
                                  /* if this is single byte IE */
                                  if (ie_list[ie_len] & 0x80)
                                      /* increment by one byte */
                                      ie_len = ie_len + 1;
                                  else
                                      /* otherwise increment by length of the IE */
                                      ie_len = ie_len + ie_list[ie_len + 1];
                              }
                          }
                      break;
                        .
                        .
                  }
       }
```

3.1.24 Non-Call Associated Signaling on Dialogic® Springware Boards (Synchronous Mode)

Non-Call Associated Signaling (NCAS) allows users to communicate by user-to-user signaling without setting up a circuit-switched connection (this signaling does not occupy B channel bandwidth). A temporary signaling connection is established (and cleared) in a manner similar to the control of a circuit-switched connection. The NCAS feature is supported for 4ESS, 5ESS, CTR4, and QSIG protocols.

Since NCAS calls are not associated with any B channel, applications should receive and transmit NCAS calls on the D channel. For T1 interfaces, this is channel 24, that is, dtiB#T24. For E1 interfaces, there is no channel (dtiB#T#) that corresponds to a D channel line device; therefore, NCAS calls (identified by the Bearer Capabilities IE content) are automatically associated with the D channel internally on dtiB#T30. Once the NCAS connection is established, the application can transmit user-to-user messages using the CRN associated with the NCAS call. The Dialogic® software and firmware support 16 simultaneous NCAS calls per D channel.

Caution: When using Dialogic® Springware Boards and T1 interfaces, attempting to perform any routing functionality on a D channel device handle (T24) causes the D channel to be brought down. For E1 interfaces, there is no equivalent issue since there is no device handle associated with the D channel.

Figure 32, Figure 33, and Figure 34 provide scenario diagrams that illustrate the operation of the NCAS feature. The NCAS scenarios are shown in Figure 35, Figure 36, Figure 37, and Figure 38.

Figure 32. User-Accepted Network-Initiated NCAS Request

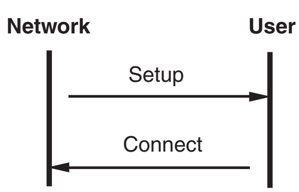


Figure 33. User-Rejected Network-Initiated NCAS Request

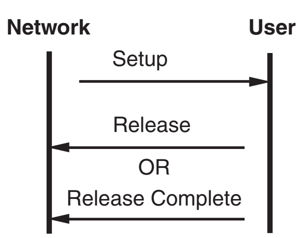
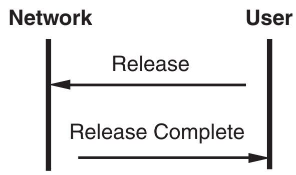


Figure 34. User-Disconnected NCAS Request



3.1.24.1 User-Initiated Call

In the following scenario, the user initiates and disconnects the NCAS call for dtiB1.

Figure 35. User-Initiated Call

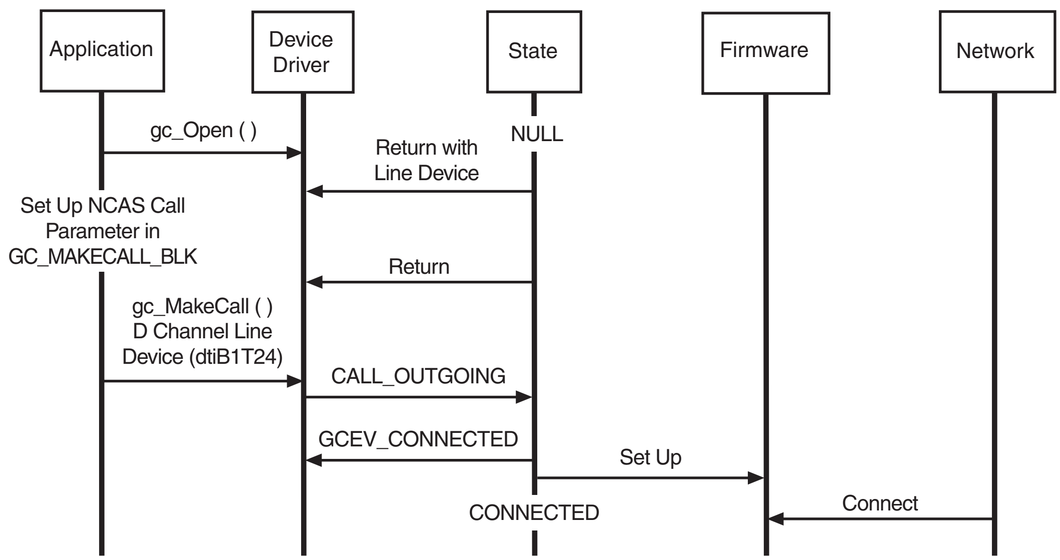
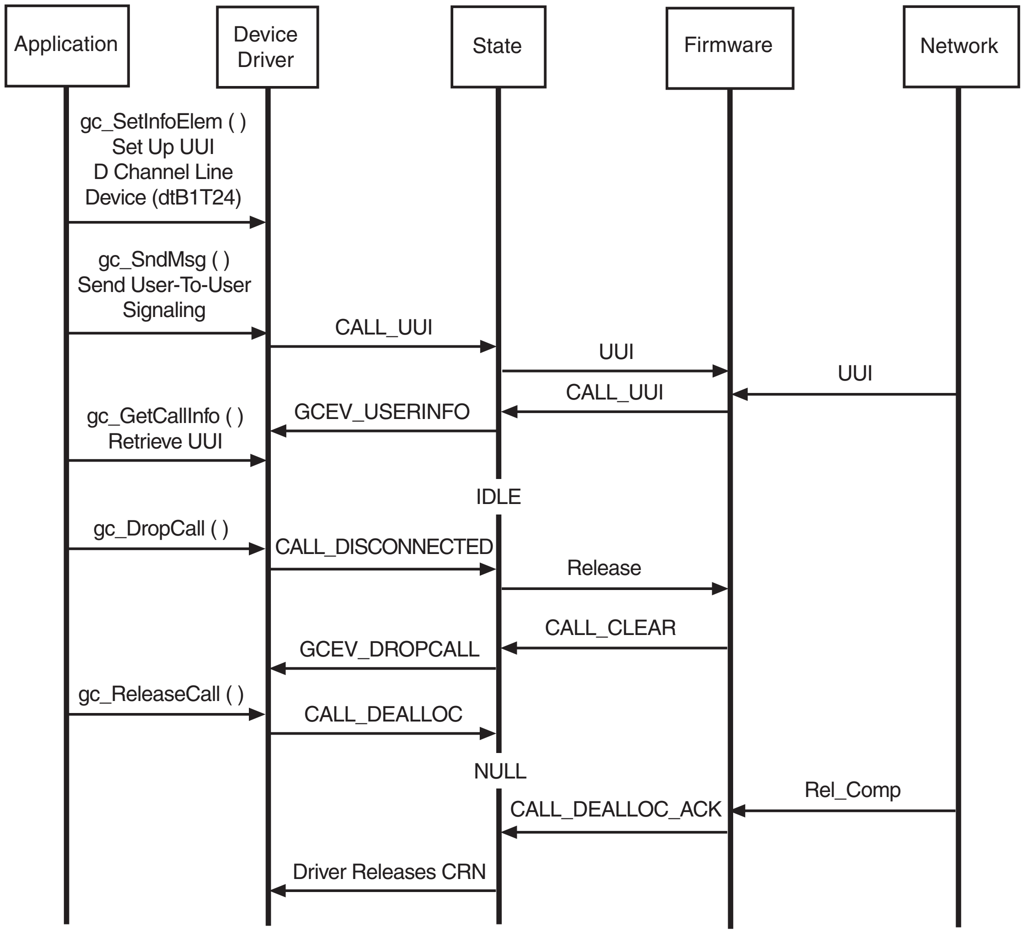


Figure 36. User-Initiated NCAS Call Connected



Example Code

The following example code illustrates the use of the Dialogic® Global Call API at various stages of the NCAS call scenario:

1. Opening a D channel line level device.

```
LINEDEV     D_chan_dev1_hdl;
      .
      .
rc = gc_Open( &D_chan_dev1_hdl, ":N_dtiB24:P_isdn", 0);
      .
```

2. Setting up the MAKECALL_BLK for NCAS call.

```
MAKECALL_BLK *makecallp;
      .
      .
// initialize makecall block
      makecallp->isdn.BC_xfer_cap             =  BEAR_CAP_UNREST_DIG;
      makecallp->isdn.BC_xfer_mode            =  ISDN_ITM_CIRCUIT;
      makecallp->isdn.BC_xfer_rate            =  PACKET_TRANSPORT_MODE;
```

```
        makecallp->isdn.usrinfo_layer1_protocol  =  NOT_USED;
        makecallp->isdn.usr_rate                 =  NOT_USED;
        makecallp->isdn.destination_number_type  =  NAT_NUMBER;
        makecallp->isdn.destination_number_plan  =  ISDN_NUMB_PLAN;
        makecallp->isdn.destination_sub_number_type     =  OSI_SUB_ADDR;
        makecallp->isdn.destination_sub_phone_number[0] ='1234'
        makecallp->isdn.origination_number_type  = NAT_NUMBER;
        makecallp->isdn.origination_number_plan  = ISDN_NUMB_PLAN;
        makecallp->isdn.origination_phone_number[0]   = '19739903000'
        makecallp->isdn.origination_sub_number_type   = OSI_SUB_ADDR;
        makecallp->isdn.origination_sub_phone_number[0] = '5678'
        makecallp->isdn.facility_feature_service        =  ISDN_SERVICE;
        makecallp->isdn.facility_coding_value    =  ISDN_SDN;
// or ISDN_ACCUNET, please check with your service provider
        makecallp->isdn.usrinfo_bufp             =  NULL;
        makecallp->isdn.nsfc_bufp                =  NULL;
              .
              .
```

3.1.24.2 Network-Initiated Call

In the following scenario, the network initiates and disconnects the NCAS call for dtiB1.

Figure 37. Network-Initiated Call

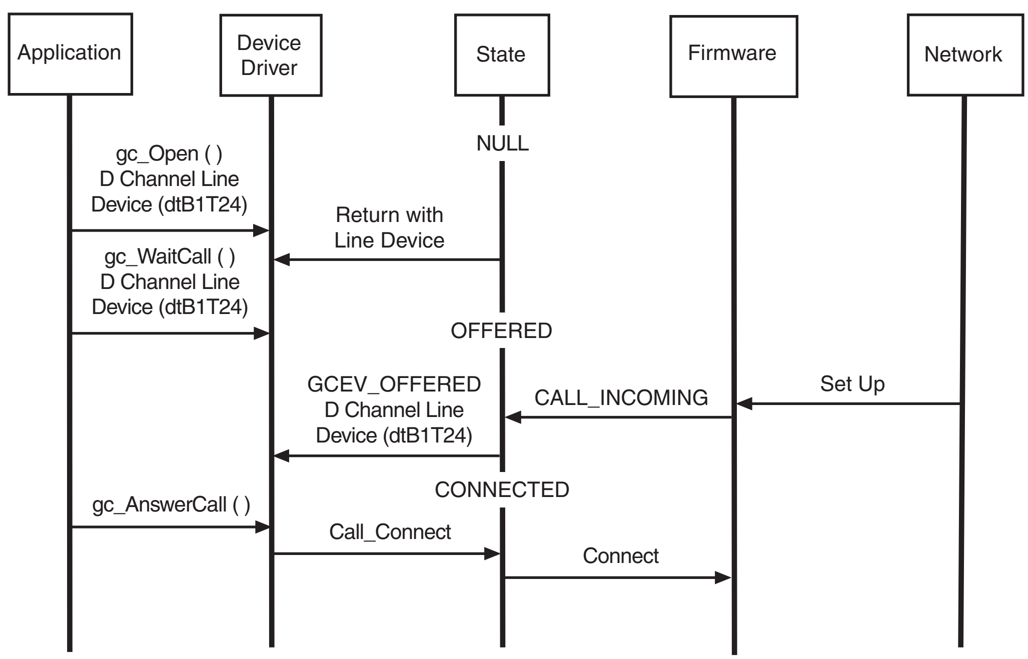
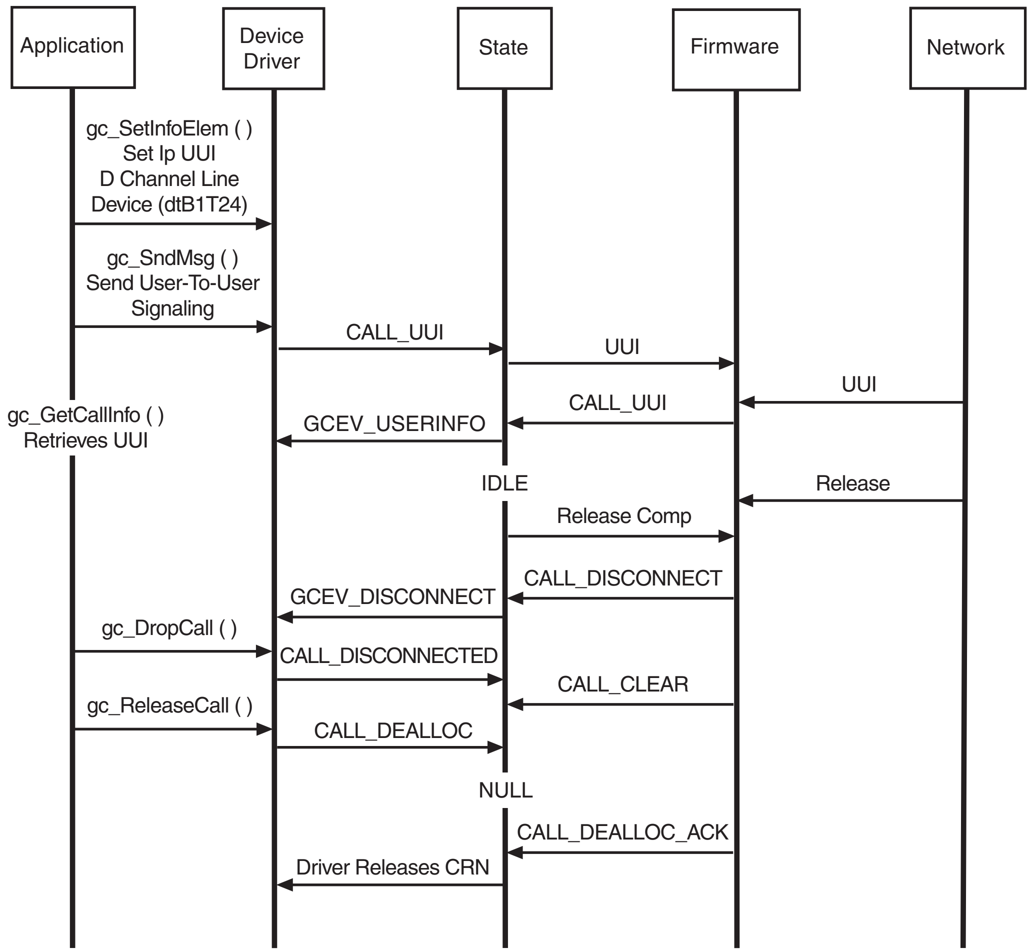


Figure 38. Network-Initiated NCAS Call Connected



Example Code

The following example code illustrates the use of the Dialogic® Global Call API to open a D channel line level device in the preceding NCAS call scenario.

```
LINEDEV     D_chan_dev1_hdl;
         .
         .
rc = gc_Open( &D_chan_dev1_hdl, ":N_dtiB24:P_isdn", 0);
         .
```

3.1.25 Non-Call Associated Signaling on Dialogic® DM3 Boards

NCAS can establish a virtual call within the network without actually associating the B channel with the call. The call only exists on the D channel, which is normally used for signaling. Once this virtual connection has been established, the customer premise equipment (CPE) can send Facility messages to the switch or terminal equipment (TE) to convey additional information. For example, Message Waiting Indicator (MWI) supplementary service information can be encoded in a Facility IE and sent in a Q.931 Setup message. The application is responsible for encoding/decoding the Facility IE.

Note: The feature is only supported on media loads that use the QSIG T1 or E1 protocol, for example, ml2_qs2_qsige1.

NCAS allows users to communicate by user-to-user signaling without setting up a circuit-switched connection. This signaling does not occupy B channel bandwidth. A temporary signaling connection is established (and cleared) in a manner similar to the control of a circuit-switched connection.

Applications must use a specific channel for NCAS calls. For E1 interfaces, this is channel 30, that is, dtiBxT30. For T1 interfaces, this is channel 23, that is, dtiBxT23.

For outbound calls, when the call is set up with the Bearer Capabilities IE indicating that it is an NCAS call, the call is sent out on the D channel, without an associated B channel. Once the NCAS connection is established, the application can transmit user-to-user messages using the call reference number (CRN) associated with the NCAS call.

For inbound calls, the Dialogic® software provides the ability to detect if the incoming call is an NCAS call or a standard circuit switched call by analyzing the information associated with the GCEV_OFFERED event triggered by the incoming call.

With Dialogic® DM3 Boards, the Dialogic® software and firmware support 8 NCAS calls per span, that is, 32 simultaneous NCAS calls per quad-span board. The 8 NCAS calls per span are *in addition* to the normal calls that you can have. For example, with T1, you can have 23 calls per span (including one on dtiBxT23), *plus* 8 NCAS calls on dtiBxT23 at the same time.

The following are notable differences between the NCAS implementation on Dialogic® Springware Boards and the NCAS implementation on Dialogic® DM3 Boards:

	Dialogic® DM3 Boards	Dialogic® Springware Boards
Channel used to make NCAS calls for T1 spans	23 (dtiBxT23)	24 (dtiBxT24)
Channel used to make NCAS calls for E1 spans	30 (dtiBxT30)	30 (dtiBxT30)
Number of simultaneous NCAS calls per D channel	8 (32 total for board with 4 spans)	16

For outbound calls, the following parameter IDs are defined in the GCIS_SET_BEARERCHNL parameter set for making NCAS calls on Dialogic® DM3 Boards:

GCIS_PARM_CODINGSTANDARD
: Set to ISDN_CODINGSTD_INTL or ISDN_CODINGSTD_CCITT.

GCIS_PARM_TRANSFERCAP
: Set to BEAR_CAP_UNREST_DIG.

GCIS_PARM_TRANSFERMODE
: Set to ISDN_ITM_CIRCUIT.

GCIS_PARM_TRANSFERRATE
: Set to PACKET_TRANSFER_MODE.

For inbound calls, the following parameter ID in the GCIS_SET_CALLTYPE parameter set is defined for receiving NCAS calls on Dialogic® DM3 Boards:

GCIS_PARM_CALL_TYPE
: Set to CALLTYPE_NCAS (to indicate an NCAS call) or CALLTYPE_CIRCUIT (to identify a standard circuit-switched call).

3.1.25.1 Outbound QSIG NCAS Call Scenarios

After opening the channel (T23 or T30), the **gc_util_insert_parm_val( )** function must be called to set up the parameters in the GCIS_SET_BEARERCHNL parameter set. For example:

```
gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_CODINGSTANDARD, sizeof(int),
ISDN_CODINGSTD_INTL);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERCAP, sizeof(int),
BEAR_CAP_UNREST_DIG);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERMODE, sizeof(int),
ISDN_ITM_CIRCUIT);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERRATE, sizeof(int),
PACKET_TRANSFER_MODE);
```

The application must also build the Facility IE (for example, with MWI information) using **gc_SetInfoElem( )** before making the call using **gc_MakeCall( )**.

Figure 39 shows the API sequence for an MWI activation with connect scenario.

Figure 39. MWI Activation with Connect Scenario

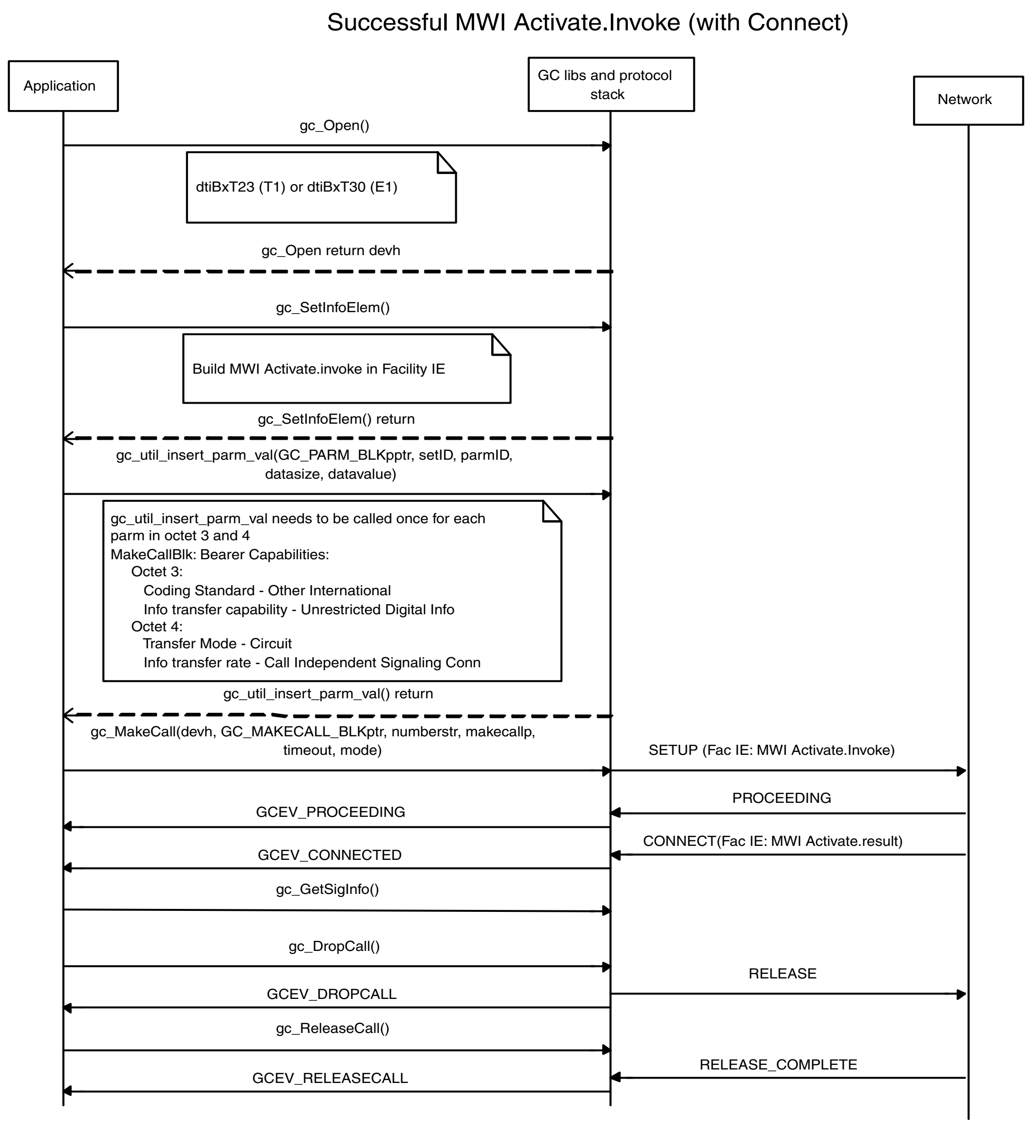


Figure 40 shows the API sequence for an MWI activation without connect scenario.

Figure 40. MWI Activation without Connect Scenario

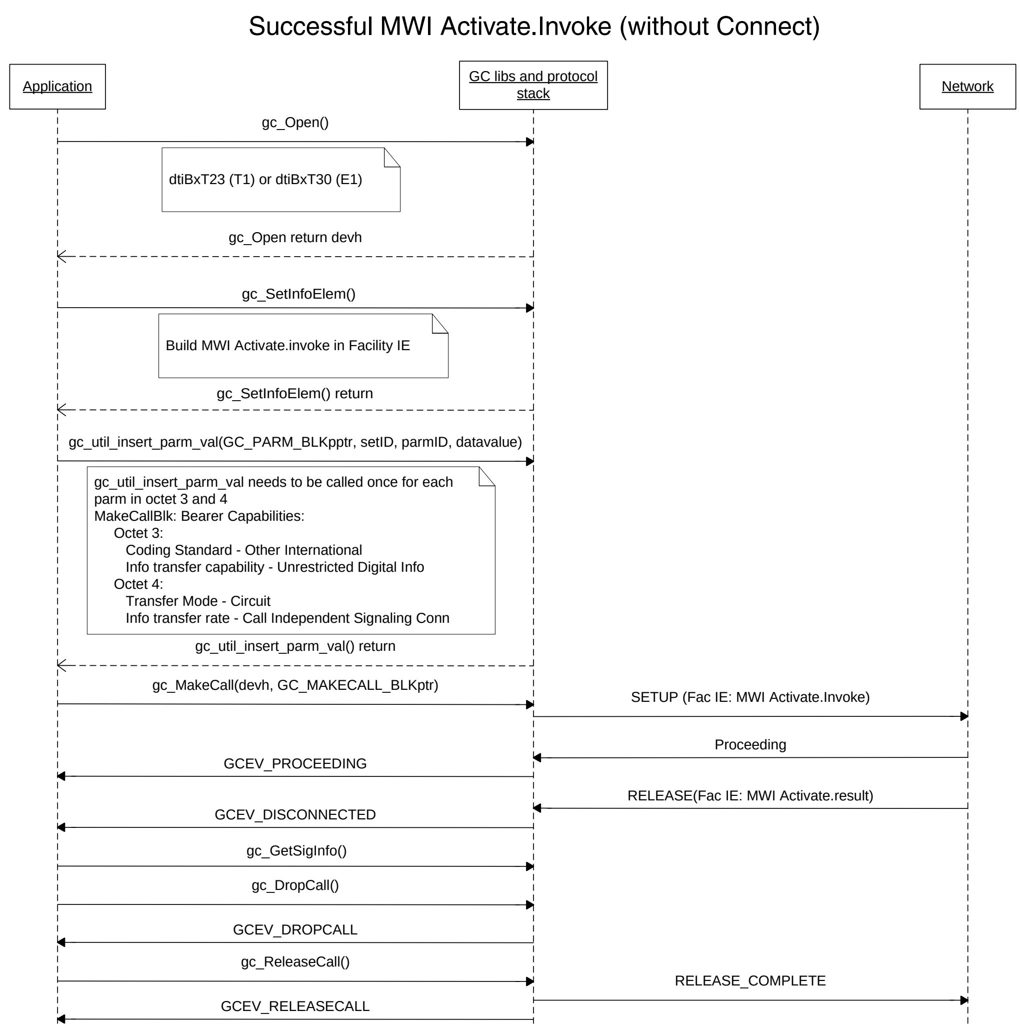


Figure 41 shows the API sequence for an MWI deactivation with connect scenario.

Figure 41. MWI Deactivation with Connect Scenario

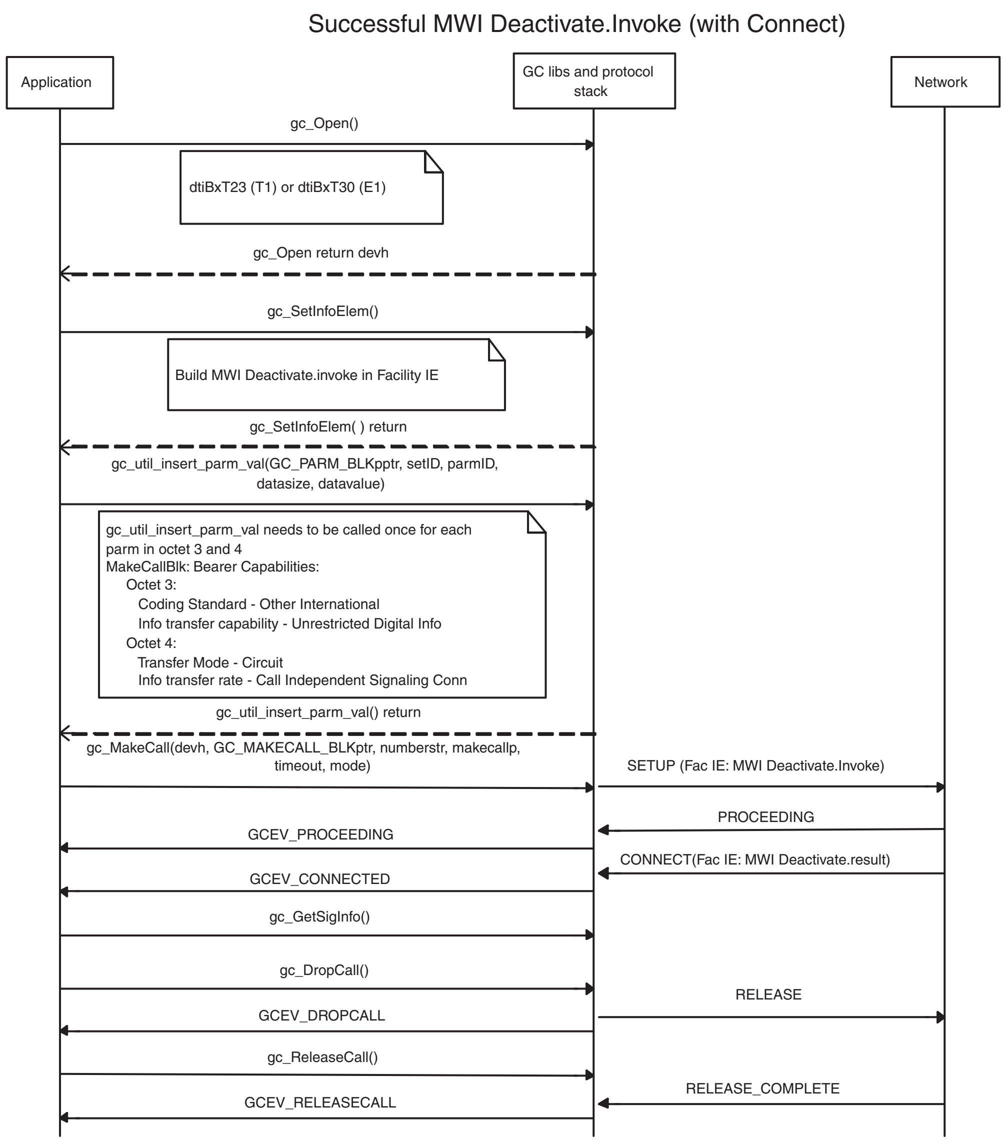
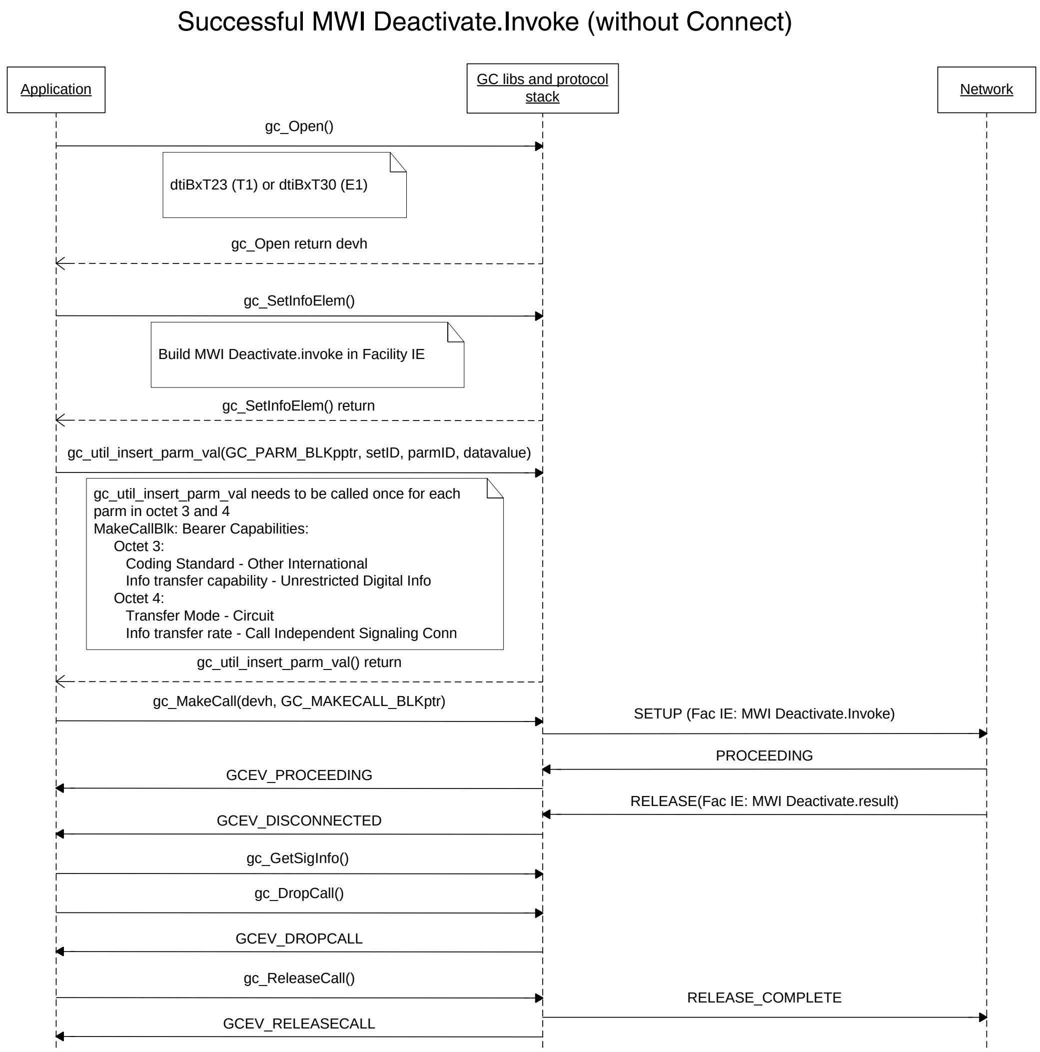


Figure 42 shows the API sequence for an MWI deactivation without connect scenario.

Figure 42. MWI Deactivation without Connect Scenario



3.1.25.2 Inbound QSIG NCAS Call Detection Scenarios

Figure 43 shows the API sequence for an MWI interrogate with connect scenario.

Figure 43. MWI Interrogate with Connect Scenario

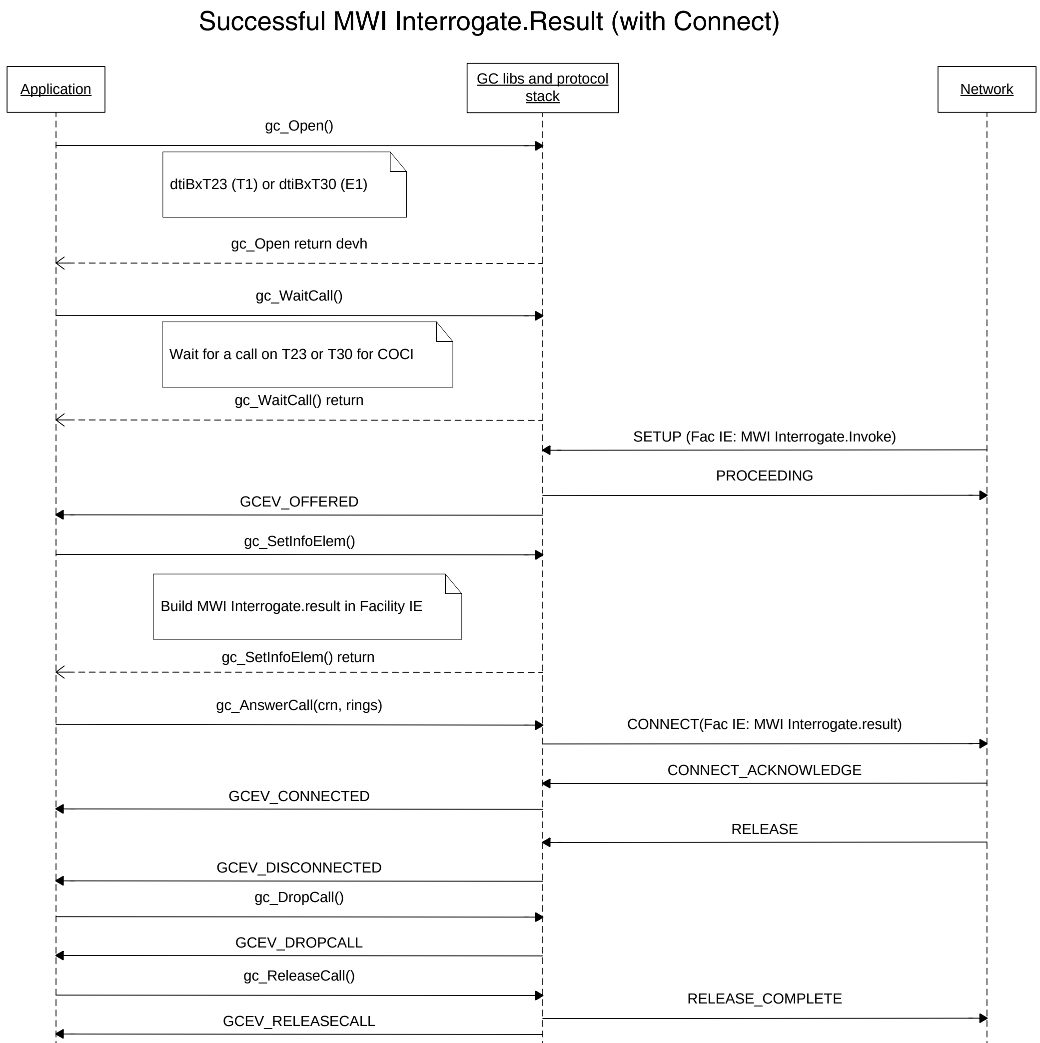
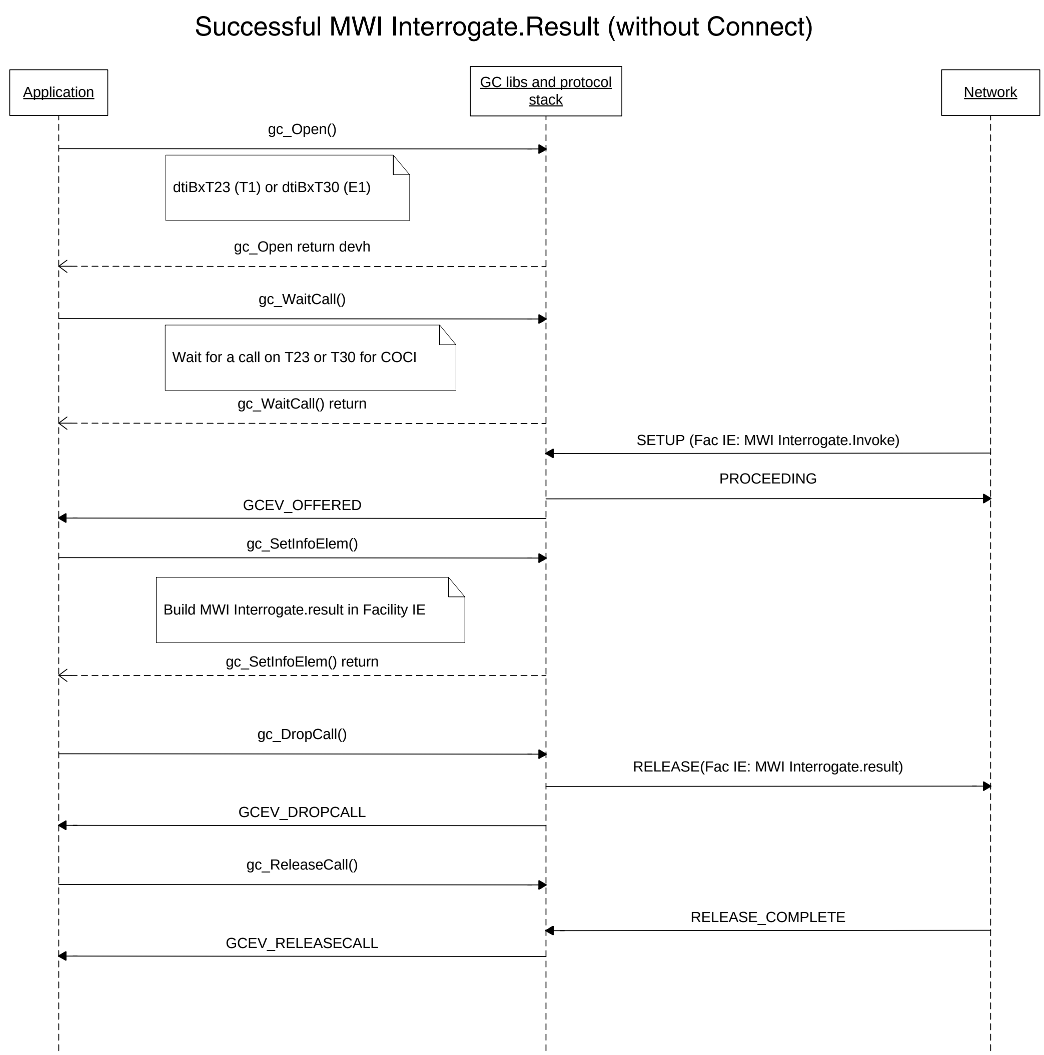


Figure 44 shows the API sequence for an MWI interrogate without connect scenario.

Figure 44. MWI Interrogate without Connect Scenario



3.1.25.3 Inbound QSIG NCAS Call Detection Code Example

The following code segment demonstrates how to retrieve the call type from the GCEV_OFFERED event triggered by an incoming call, to determine if the call is an NCAS call or a standard circuit-switched call.

```
case GCEV_OFFERED:
    GC_PARM_BLKP gcParmBlkp = NULL;
    GC_PARM_DATAP t_gcParmDatap = NULL;
    EXTENSIONEVTBLK *ext_evtblkp = NULL;

    ext_evtblkp = (EXTENSIONEVTBLK *)meta_event.extevtdatap;
    gcParmBlkp = &ext_evtblkp->parmblk;
    while (t_gcParmDatap = gc_util_next_parm(gcParmBlkp, t_gcParmDatap))
    {
        switch (t_gcParmDatap->set_ID)
        {
            case GCIS_SET_CALLTYPE:
                switch(t_gcParmDatap->parm_ID)
                {
                    case GC_PARM_CALL_TYPE:
                    // Determine the Call Type.
                    switch (t_gcParmDatap->value_buf)
                    {
                        case CALLTYPE_NCAS:
                            cout << "NCAS call detected" << endl;
                            break;
                        case CALLTYPE_CIRCUIT:
                            cout << "Regular call detected" << endl;
                            break;
                    }
                break;

            default:
                cout << "Unknown PARM ID" << endl;
                break;
            }
        break;

        default:
            cout << "Unknown SET ID" << endl;
            break;
        }
    }
    break;
```

3.1.26 Call Hold and Retrieve Scenarios

When using Dialogic® DM3 Boards, the following scenarios describe the Dialogic® Global Call API functions and events used when implementing call hold and retrieve functionality:

- Call Hold at the Holding Side
- Call Hold at the Held Side
- Call Retrieve at the Holding Side
- Call Retrieve at the Held Side

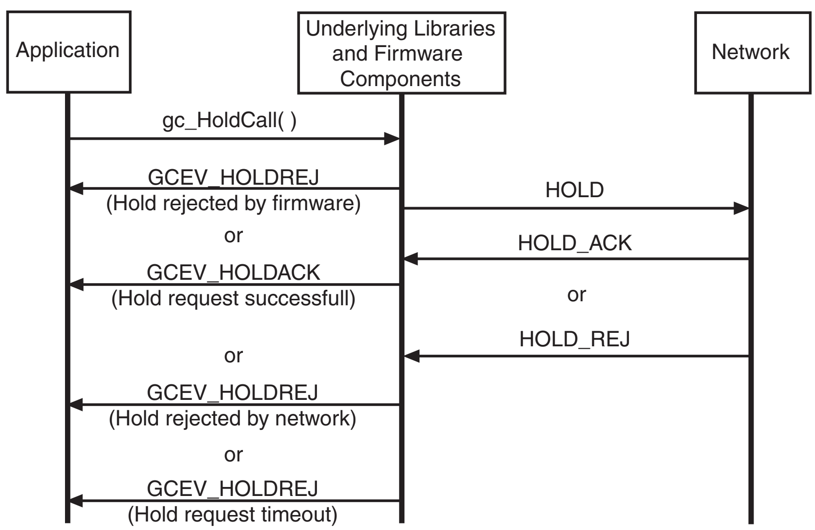
Notes: ***1.*** Call hold and retrieve is supported in the Proceeding, Accepted, Alerting, and Connected states. The ability to do a second **gc_MakeCall( )** in any of these states is supported.

2. When using Dialogic® DM3 Boards, call hold and retrieve as described in this section is supported for the 4ESS, 5ESS, DMS, NET5, NI2, NTT, and QSIG protocols.

3.1.26.1 Call Hold at the Holding Side

Figure 45 shows the sequence of function calls and events at the holding side when putting a call on hold.

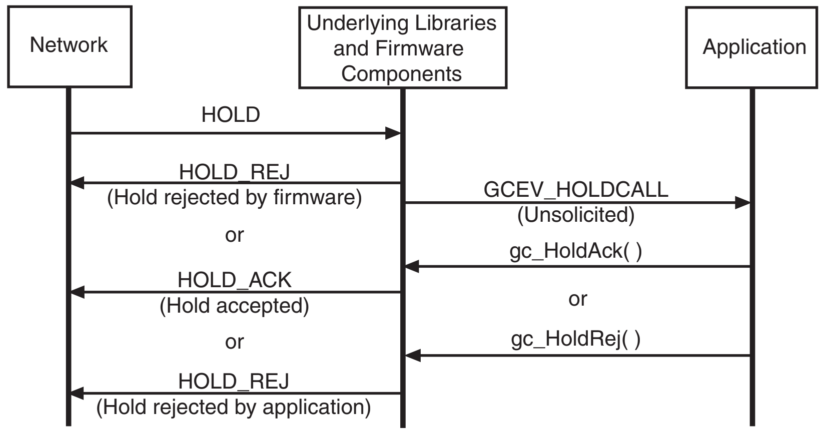
Figure 45. Call Hold Scenario at the Holding Side



3.1.26.2 Call Hold at the Held Side

Figure 46 shows the sequence of function calls and events at the held side when putting a call on hold.

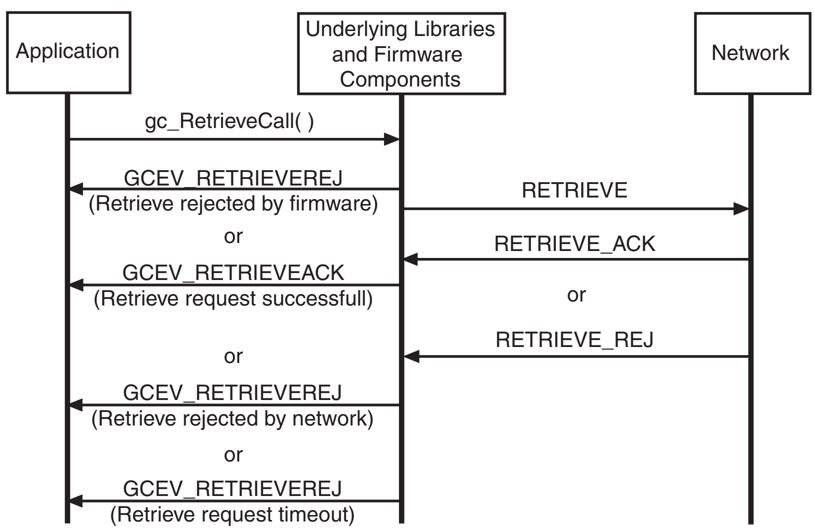
Figure 46. Call Hold Scenario at the Held Side



3.1.26.3 Call Retrieve at the Holding Side

Figure 47 shows the sequence of function calls and events at the holding side when retrieving a call from the hold state.

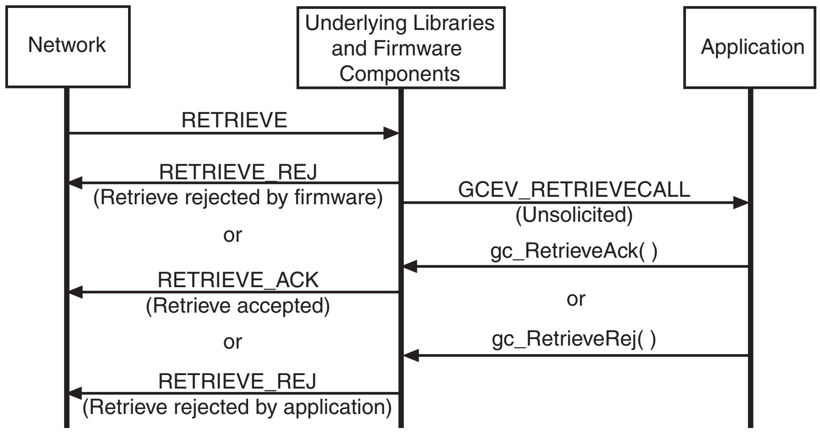
Figure 47. Call Retrieve Scenario at the Holding Side



3.1.26.4 Call Retrieve at the Held Side

Figure 48 shows the sequence of function calls and events at the held side when retrieving a call from the hold state.

Figure 48. Call Retrieve Scenario at the Held Side



3.2 DPNSS-Specific Call Scenarios

Call scenarios that are specific to the DPNSS protocol are described in this section. Each scenario includes:

- A table that illustrates the Dialogic® Global Call API functions issued by the application to either initiate a transaction or to respond to an external action, and the resulting events that are returned to the application.
- A step-by-step description of the scenario following each table.

The following call control scenarios are described:

- Executive Intrusion
- Executive Intrusion with Prior Validation
- Locally Initiated Hold and Retrieve
- Remotely Initiated Hold and Retrieve
- Local Diversion at the Outbound Side
- Local Diversion at the Inbound Side
- Remote Diversion at the Outbound Side
- Remote Diversion at the Inbound Side
- Call Transfer
- Virtual Call at the Outbound Side
- Virtual Call at the Inbound Side

3.2.1 Executive Intrusion

Table 11 describes the DPNSS call scenario.

Table 11. DPNSS Executive Intrusion Scenario

Step	API	Action/Result	Event
1	**gc_MakeCall( )** (with Intrusion IE)	--->	
2		<---	GCEV_PROCEEDING
3	---	Intrusion succeeded <---	--- GCEV_CONNECTED
4	---	Intrusion failed <---	--- GCEV_DISCONNECTED

The procedure is as follows:

1. The application places an outgoing call using the **gc_MakeCall( )** function to a busy extension with Intrusion Type set to the INTRUDE_NORMAL value. See Table 68, "Intrusion IE", on page 304 for the format of the Intrusion IE.
2. Receives call proceeding (GCEV_PROCEEDING) event.
3. Receives call connected (GCEV_CONNECTED) event. Call successfully intruded.
4. Receives call disconnected (GCEV_DISCONNECTED) event. Call was not intruded.

3.2.2 Executive Intrusion with Prior Validation

Table 12 describes the DPNSS call scenario.

Table 12. DPNSS Executive Intrusion with Prior Validation Scenario

Step	API	Action/Result	Event
1	**gc_MakeCall( )** (with Intrusion IE)	--->	
2		<---	GCEV_PROCEEDING (with Busy IE)
3	**gc_SndMsg( )** (SndMsg_Intrude)	--->	
4	---	Intrusion succeeded <---	--- GCEV_CONNECTED
5	---	Intrusion failed <---	--- GCEV_DISCONNECTED

The procedure is as follows:

1. Application places an outgoing call using the **gc_MakeCall( )** function to a busy extension with Intrusion Type set to the INTRUDE_PRIOR_VALIDATION value. See Table 68, "Intrusion IE", on page 304 for the format of the Intrusion IE.
2. Receives call proceeding (GCEV_PROCEEDING) event with indication that the remote party was busy. Use the **gc_GetSigInfo( )** function to retrieve the BUSY_IE value. See Table 60, "Intrusion IE", on page 301 for the format of the Busy IE.
3. Application sends intrude request using the **gc_SndMsg( )** function. See the **gc_SndMsg( )** function description in the *Dialogic® Global Call API Library Reference* and Section 8.2.38, "gc_SndMsg( ) Variances for ISDN", on page 245 for ISDN-specific information.
4. Receives call connected (GCEV_CONNECTED) event. Call successfully intruded.
5. Receives call disconnected (GCEV_DISCONNECTED) event. Call was not intruded.

3.2.3 Locally Initiated Hold and Retrieve

Table 13 describes the DPNSS call scenario.

Table 13. DPNSS Locally Initiated Hold and Retrieve Scenario

Step	API	Action/Result	Event
1	--- **gc_HoldCall( )**	Call Connected --->	---
2	---	Call Held <---	--- GCEV_HOLDACK
3	Unroute SCbus time slot for held call	--->	
4	**gc_RetrieveCall( )**		
5		<---	GCEV_RETRIEVEACK
6	Reroute SCbus time slot for retrieved call		
7	---	Call not held <---	--- GCEV_HOLDREJ
8	Take no action		

The procedure is as follows:

1. The application places a connected call on hold using the **gc_HoldCall( )** function.
2. When the call is held, the application will receive a hold acknowledge (GCEV_HOLDACK) event.
3. The application should unroute the SCbus time slot for the held call.
4. The application retrieves a held call using the **gc_RetrieveCall( )** function.
5. When the call is retrieved, the application will receive a retrieve acknowledge (GCEV_RETRIEVEACK) event.
6. The application should reroute the SCbus time slot for the retrieved call.
7. When a call is not held, the application will receive a hold reject (GCEV_HOLDREJ) event.
8. The application should take no action on the call's SCbus time slot.

Note: The retrieval of a held call cannot be rejected when using the DPNSS protocol.

3.2.4 Remotely Initiated Hold and Retrieve

Table 14 describes the DPNSS call scenario.

Table 14. DPNSS Remotely Initiated Hold and Retrieve Scenario

Step	API	Action/Result	Event
1	---	Call Connected <---	--- GCEV_HOLDCALL
2	--- Unroute SCbus time slot for held call	Call Held	---
3	**gc_HoldAck( )**	--->	
4		<---	GCEV_RETRIEVECALL
5	Reroute SCbus time slot for retrieved call		
6	--- Take no action	Call not held	---
7	**gc_HoldRej( )**	--->	

The procedure is as follows:

1. A request (GCEV_HOLDCALL event) to place a connected call on hold is received.
2. The application accepts the hold request and should unroute the SCbus time slot for the requested call.
3. The application accepts the hold request using the **gc_HoldAck( )** function.
4. A request (GCEV_RETRIEVECALL event) to retrieve a held call is received.
5. The application receives the retrieve request and should reroute the SCbus time slot for the requested call.
6. The application rejects the hold request and takes no action on the call's SCbus time slot.
7. The application rejects the hold request using the **gc_HoldRej( )** function.

Note: The retrieval of a held call cannot be rejected when using the DPNSS protocol.

3.2.5 Local Diversion at the Outbound Side

Table 15 describes the DPNSS call scenario.

Table 15. DPNSS Local Diversion at the Outbound Side Scenario

Step	API	Action/Result	Event
1		<---	GCEV_OFFERED
2	**gc_SndMsg( )** (SndMsg_Divert, Diversion Location: DIVERT_LOCAL)	--->	
3	**gc_AnswerCall( )**	--->	
4		<---	GCEV_ANSWERED

The procedure is as follows:

1. An incoming call (GCEV_OFFERED) event is received.
2. The application diverts the incoming call to a different extension using the **gc_SndMsg( )** function. See the **gc_SndMsg( )** function description in the *Dialogic® Global Call API Library Reference* and Section 8.2.38, "gc_SndMsg( ) Variances for ISDN", on page 245 for ISDN-specific information.
3. The application answers the call using the **gc_AnswerCall( )** function.
4. A call answered (GCEV_ANSWERED) event is received.

3.2.6 Local Diversion at the Inbound Side

Table 16 describes the DPNSS call scenario.

Table 16. DPNSS Local Diversion at the Inbound Side Scenario

Step	API	Action/Result	Event
1	**gc_MakeCall( )**	--->	
2		<---	GCEV_PROCEEDING (with Diversion IE, diversion location: DIVERT_LOCAL)
3		<---	GCEV_CONNECTED

The procedure is as follows:

1. The application places an outgoing call using the **gc_MakeCall( )** function.
2. A call proceeding (GCEV_PROCEEDING) event with an indication that the call was diverted to another location is received. Use the **gc_GetSigInfo( )** function to retrieve the Diversion IE. See Table 69, “Diversion IE”, on page 304 for the Diversion IE format.
3. A call connected (GCEV_CONNECTED) event is received and the call is established.

3.2.7 Remote Diversion at the Outbound Side

Table 17 describes the DPNSS call scenario.

Table 17. DPNSS Remote Diversion at the Outbound Side Scenario

Step	API	Action/Result	Event
1	**gc_MakeCall( )**	--->	
2	**gc_SndMsg( )** (SndMsg_Divert, Diversion Location: DIVERT_REMOTE)	<---	GCEV_PROCEEDING (with Diversion IE, Diversion Location: DIVERT_REMOTE)
3	**gc_DropCall( )**	--->	
4		<---	GCEV_DROPCALL
5	**gc_ReleaseCall( )**	--->	
6	**gc_MakeCall( )** (with Diversion IE)	--->	
	---	Divert achieved	---
7		<---	GCEV_PROCEEDING
8		<---	GCEV_DIVERTED
9		<---	GCEV_CONNECTED
	---	Divert failed	---
10		<---	GCEV_DISCONNECTED

The procedure is as follows:

1. Party 1 calls Party 2 by issuing the **gc_MakeCall( )** function.
2. Party 1 receives a GCEV_PROCEEDING event from Party 2 with an indication that the call needs to be diverted to Party 3. The Diversion IE will contain the telephone number of Party 3. See Table 69, “Diversion IE”, on page 304 for the Diversion IE format.
3. Party 1 disconnects original call to Party 2 using a **gc_DropCall( )** function.
4. Party 1 receives a call disconnect (GCEV_DROPCALL) event from Party 2.
5. The application releases the first call using a **gc_ReleaseCall( )** function.
6. Party 1 diverts the call to Party 3 by issuing a **gc_MakeCall( )** function. Calling party number IE should contain Party 3's telephone number. Diversion IE should contain Party 2's telephone number. See the **gc_SetUserInfo( )** function description in the *Dialogic® Global Call API Library Reference* and Section 8.2.36, “gc_SetUserInfo( ) Variances for ISDN”, on page 244 for ISDN-specific information.
7. Party 1 receives a proceeding (GCEV_PROCEEDING) event from Party 3.
8. Party 1 receives a divert successful (GCEV_DIVERTED) event from Party 3.
9. Party 1 receives a call connected (GCEV_CONNECTED) event from Party 3. The call is successfully diverted.
10. Party 1 receives a divert failed (GCEV_DISCONNECTED) event from Party 3. The call was not diverted.

3.2.8 Remote Diversion at the Inbound Side

Table 18 describes the DPNSS call scenario.

Table 18. DPNSS Remote Diversion at the Inbound Side Scenario

Step	API	Action/Result	Event
1		<---	GCEV_OFFERED
2	**gc_SndMsg( )** (SndMsg_Divert, Diversion Location: DIVERT_REMOTE)	--->	
3		<---	GCEV_DISCONNECTED
4	**gc_DropCall( )**	--->	
5		<---	GCEV_DROPCALL
6	**gc_ReleaseCall( )**	--->	

The procedure is as follows:

1. Party 2 receives an incoming call (GCEV_OFFERED) event from Party 1.
2. Party 2 diverts incoming call to Party 3. Send Party 3's telephone number as Diversion number. See Table 76, "SndMsg_Divert", on page 306 for the format of the SndMsg_Divert message.
3. Party 1 disconnects call to Party 2.
4. Party 2 drops call using the **gc_DropCall( )** function.
5. Party 2 receives a drop call (GCEV_DROPCALL) event from Party 1.
6. Party 2 releases the call using the **gc_ReleaseCall( )** function.

3.2.9 Call Transfer

Table 19 describes the DPNSS call scenario.

Table 19. DPNSS Call Transfer Scenario

Step	API	Action/Result	Event
1		<---	GCEV_OFFERED (CRN 1)
2	**gc_AnswerCall( )** (CRN 1)	--->	
3	**gc_HoldCall( )** (CRN 1)	--->	
4		<---	GCEV_HOLDACK (CRN 1)
5	**gc_MakeCall( )**	--->	
6		<---	GCEV_PROCEEDING (CRN 2 with Inquiry IE)
7		<---	GCEV_CONNECTED (CRN 2 with Inquiry IE)
8	**gc_SndMsg( )** (SndMsg_Transfer, CRN 1)	--->	
9	**gc_SndMsg( )** (SndMsg_Transfer, CRN 2)	--->	
10			GCEV_TRANSFERACK (CRN 1)
11			GCEV_TRANSFERACK (CRN 2)
12	Cross connect CRN 1's and CRN 2's SCbus time slots		
13		<---	GCEV_TRANSIT (CRN 1)
14	**gc_SndMsg( )** (SndMsg_Transit, CRN 2)	--->	
15		<---	GCEV_TRANSIT (CRN 2)
16	**gc_SndMsg( )** (SndMsg_Transit, CRN 1)	--->	
17		<---	GCEV_DISCONNECTED (CRN 1)
18	**gc_DropCall( )** (CRN 1)	--->	
19		<---	GCEV_DROPCALL (CRN 1)
20	**gc_ReleaseCall( )** (CRN 1)	--->	
21		<---	GCEV_DISCONNECTED (CRN 2)
22	**gc_DropCall( )** (CRN 2)	--->	
23		<---	GCEV_DROPCALL (CRN 2)
24	**gc_ReleaseCall( )** (CRN 2)	--->	

The procedure is as follows:

1. Party 2 receives an incoming call (GCEV_OFFERED) event from Party 1.
2. Party 2 answers call from Party 1 using the **gc_AnswerCall( )** function.
3. Party 2 places the call on hold using the **gc_HoldCall( )** function.

   Some switches may not support holding a call.
4. Party 2 receives a call on hold acknowledge (GCEV_HOLDACK) event.
5. Party 2 places an inquiry call to Party 3 using the **gc_MakeCall( )** function. The application should use Party 1's telephone number as the calling party number and Party 3's telephone number as called party number. See Table 71, "Inquiry IE", on page 305 for the Inquiry IE format.
6. Party 2 receives a call proceeding (GCEV_PROCEEDING) event with an Inquiry IE from Party 3. See Table 71, "Inquiry IE", on page 305 for the Inquiry IE format.
7. Party 2 receives a call connected (GCEV_CONNECTED) event with Inquiry IE from Party 3. See Table 71, "Inquiry IE", on page 305 for the Inquiry IE format.
8. Party 2 sends a transfer request to Party 1 with a TRANSFER_ORIG value as the transfer direction using the **gc_SndMsg( )** function. See Table 79, "SndMsg_Transfer", on page 307 for the message format.
9. Party 2 sends a transfer request to Party 3 with a TRANSFER_TERM value as the transfer direction using the **gc_SndMsg( )** function. See Table 79, "SndMsg_Transfer", on page 307 for the message format.
10. Party 2 receives a transfer acknowledge (GCEV_TRANSFERACK) event from Party 1.
11. Party 2 receives a transfer acknowledge (GCEV_TRANSFERACK) event from Party 3. Transfer completed. At this time, Party 2 loses control of the call.
12. The application should cause Party 1 to listen to Party 2's SCbus transmit time slot and Party 2 to listen to Party 1's SCbus transmit time slot.
13. Party 2 receives a transit (GCEV_TRANSIT) event from Party 1. Party 2 should retrieve the content of the Transit IE using the **gc_GetSigInfo( )** function.
14. Party 2 sends content of the Transit IE (unchanged) from Party 1 to Party 3 using the **gc_SndMsg( )** function. See Table 80, "SndMsg_Transit", on page 307 for the message format.
15. Party 2 receives a transit (GCEV_TRANSIT) event from Party 3. Party 2 should retrieve the content of the Transit IE using the **gc_GetSigInfo( )** function.
16. Party 2 sends content of Transit IE (unchanged) from Party 3 to Party 1 using the **gc_SndMsg( )** function. See Table 80, "SndMsg_Transit", on page 307 for the message format.
17. Party 2 receives a disconnect all (GCEV_DISCONNECTED) event from Party 1.
18. Party 2 drops the call to Party 1 using the **gc_DropCall( )** function.
19. Party 2 receives a drop call (GCEV_DROPCALL) event from Party 1.
20. Party 2 releases the call to Party 1 using the **gc_ReleaseCall( )** function.
21. Party 2 receives a disconnect call (GCEV_DISCONNECTED) event from Party 3.
22. Party 2 drops the call to Party 3 using the **gc_DropCall( )** function.
23. Party 2 receives a drop call (GCEV_DROPCALL) event from Party 3.

24. Party 2 releases call to Party 3 using the **gc_ReleaseCall( )** function.

Notes: ***1.*** Steps 3 and 4 are optional and need not be carried out on most PBXs.

2. Steps 12 through 16 may be repeated multiple times depending on when or whether the distant PBX supports Route Optimization. When Route Optimization occurs, or if either end of the transferred call is terminated, the call flow proceeds to step 17.

3.2.10 Virtual Call at the Outbound Side

Table 20 describes the DPNSS call scenario.

Table 20. DPNSS Virtual Call at the Outbound Side Scenario

Step	API	Action/Result	Event
1	**gc_MakeCall( )** (with Virtual Call IE)	--->	
2		<---	GCEV_DISCONNECTED
3	**gc_DropCall( )**	--->	
4		<---	GCEV_DROPCALL
5	**gc_ReleaseCall( )**	--->	

The procedure is as follows:

1. The application places an outgoing call with Virtual Call IE and any other information set, such as NSI strings or Extension Status, using the **gc_MakeCall( )** function. See Table 73, "Virtual Call IE", on page 305 for the format of the Virtual Call IE.
2. The application receives a call disconnected (GCEV_DISCONNECTED) event. Use the **gc_ResultValue( )** function to retrieve the clearing cause. A RESP_TO_STAT_ENQ value means that the call was Acknowledged, and a FACILITY_REJECT value means that the call was Rejected.
3. The application issues a **gc_DropCall( )** function.
4. A drop call (GCEV_DROPCALL) event is received.
5. The application issues a **gc_ReleaseCall( )** function.

3.2.11 Virtual Call at the Inbound Side

Table 21 describes the DPNSS call scenario.

Table 21. DPNSS Virtual Call at the Inbound Side Scenario

Step	API	Action/Result	Event
1		<---	GCEV_OFFERED (with Virtual Call IE)
2	**gc_DropCall( )**	--->	
3		<---	GCEV_DROPCALL
4	**gc_ReleaseCall( )**	--->	

The procedure is as follows:

1. The application receives a call offered (GCEV_OFFERED) event with an indication that this is a virtual call. Use the **gc_GetSigInfo( )** function to retrieve the Virtual Call IE and any other information, such as NSI strings.
2. The application issues a **gc_DropCall( )** function with clearing cause set to the RESP_TO_STAT_ENQ value to acknowledge the call, or set to the FACILITY_REJECT value to reject the call.
3. A drop call (GCEV_DROPCALL) event is received.
4. The application issues a **gc_ReleaseCall( )** function.

ISDN-Specific Operations 4

This chapter describes how to perform Integrated Services Digital Network (ISDN)-specific operations while developing an application that uses ISDN technology. The operations are divided into the following categories:

4.1 Operations Performed Using FTE

The following sections describe the ISDN-specific operations that can be performed using the Dialogic® Global Call API Feature Transparency and Extension (FTE) capability, more specifically, the **gc_Extension( )** function with certain extension IDs (**ext_id**). Additional information about the required input parameters, as well as any applicable cautions and example codes are also provided. The parameter set IDs and parameter IDs that are referenced are described in Chapter 9, “ISDN-Specific Parameter Reference”. The operations that can be performed include:

- Send a Progress Message to the Network
- Retrieve the Status of the B Channel

- Retrieve the Status of the D Channel
- Retrieve the Logical Data Link State
- Retrieve the CES and SAPI (BRI Only)
- Retrieve Frame from Application
- Retrieve the Network Call Reference Value (CRV)
- Retrieve Information for a GLOBAL or NULL CRN Event
- Play a User-Defined Tone
- Set the Logical Data Link State
- Send Frame to the Data Link Layer
- Send a Non-Call State Related ISDN Message
- Send a Non-Call Related ISDN Message
- Stop Currently Playing Tone (BRI Only)
- Redefine Call Progress Tone Attributes (BRI Only)

4.1.1 Send a Progress Message to the Network

Note: The GCIS_EXID_CALLPROGRESS extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the Progress message can be sent using **gc_SndMsg( )** function.

The GCIS_EXID_CALLPROGRESS extension ID is used to send a progress message to the network. The **gc_Extension( )** API can be called with this **ext_id** after GCEV_OFFERED occurs, in asynchronous mode, or after the **gc_WaitCall( )** function successfully completes, in synchronous mode. Applications may use the message on the D channel to indicate that the connection is not an ISDN terminal or that in-band information is available.

Calling the **gc_Extension( )** function with the GCIS_EXID_CALLPROGRESS extension ID is not needed in the terminating node. It may be used in a drop-and-insert configuration when an in-band Special Information Tone (SIT) or call progress tone is sent to the network.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CRN
target_id	the call reference number (CRN) of the call
ext_id	GCIS_EXID_CALLPROGRESS

Parameter	Input
parmblkp	**set_id** – GCIS_SET_CALLPROGRESS **parm_id** – GCIS_PARM_CALLPROGRESS_INDICATOR **values** – One of the following: • CALL_NOT_END_TO_END_ISDN • IN_BAND_INFO **value_type** – int
mode	EV_SYNC

Code Example

```
#include "gclib.h"
#include "gcerr.h"
#include "gcisdn.h"

int extSndProgress(CRN crn)
{
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;
    int indicator;

    indicator = CALL_NOT_END_TO_END_ISDN;

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_CALLPROGRESS,
                 GCIS_PARM_CALLPROGRESS_INDICATOR, sizeof( int ), &indicator);

    mode = EV_SYNC;

    ret_val = gc_Extension( GCTGT_GCLIB_CRN, crn,
        GCIS_EXID_CALLPROGRESS, parm_blkp, &ret_blkp, mode);
    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);

           } else {
                    printf("gc_ErrorInfo() call failed\n");
           }
    }



    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.2 Retrieve the Status of the B Channel

Notes: **1.** The GCIS_EXID_GETBCHANSTATE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the B channel state can be retrieved using **gc_GetLineDevState**( ) function.

2. This feature is **not** supported for the Dialogic® BRI/2 Board.

The GCIS_EXID_GETBCHANSTATE extension ID is used for retrieving the status (in service, in maintenance, or out of service) of the B channel at any time.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the B channel
ext_id	GCIS_EXID_GETBCHANSTATE
parmblkp	NULL
retblkp	**set_id** – GCIS_SET_CHANSTATE **parm_id** – GCIS_PARM_BCHANSTATE **values** – One of the following: • ISDN_IN_SERVICE • ISDN_MAINTENANCE • ISDN_OUT_OF_SERVICE **value_type** – int
mode	EV_SYNC

Code Example

```
int extGetBChanState (LINEDEV handle)
{
    GC_PARM_DATAP parm_datap;
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;
    mode = EV_SYNC;

    ret_val = gc_Extension( GCTGT_GCLIB_CHAN, handle,
        GCIS_EXID_GETBCHANSTATE, parm_blkp, &ret_blkp, mode);

    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);

           } else {
                    printf("gc_ErrorInfo() call failed\n");
           }
    }

    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.3 Retrieve the Status of the D Channel

Notes: ***1.*** The GCIS_EXID_GETDCHANSTATE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the D channel state can be retrieved using **gc_GetLineDevState( )** function.

2. The GCIS_EXID_GETDCHANSTATE extension ID applies only to ISDN PRI technology. For ISDN BRI technology, use the GCIS_EXID_GETDLINKSTATE extension ID.

The GCIS_EXID_GETDCHANSTATE extension ID is used for retrieving the status of the D channel of a specified board at any time.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the D channel
ext_id	GCIS_EXID_GETDCHANSTATE
parmblkp	NULL
retblkp	**set_id** – GCIS_SET_CHANSTATE **parm_id** – GCIS_PARM_DCHANSTATE **values** – One of the following: • DATA_LINK_DOWN • DATA_LINK_UP **value_type** – int
mode	EV_SYNC

Code Example

```
int extGetDChanState (LINEDEV handle)
{
    GC_PARM_DATAP parm_datap;
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;
    mode = EV_SYNC;
    ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle, GCIS_EXID_GETDCHANSTATE, parm_blkp,
                           &ret_blkp, mode);
    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);

           } else {
                    printf("gc_ErrorInfo() call failed\n");
           }
    }
```

```
    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.4 Retrieve the Logical Data Link State

Note: The GCIS_EXID_GETDLINKSTATE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_GETDLINKSTATE extension ID is **not** supported.

The GCIS_EXID_GETDLINKSTATE extension ID is used for retrieving the logical data link state (operable, inoperable, or disabled) of the specified board device for PRI or station device for BRI.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	board device handle for PRI, station device handle for BRI
ext_id	GCIS_EXID_GETDLINKSTATE
parmblkp	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char
retblkp	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_STATE **values** – One of the following: • DATA_LINK_DOWN • DATA_LINK_UP • DATA_LINK_DISABLED **value_type** – int
mode	EV_SYNC

Code Example

```
int extGetDLinkState (LINEDEV handle)
{
    GC_PARM_DATAP parm_datap;
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
```

```
    GC_INFO  t_Info;
    char sapi, ces;
    sapi = 0;
    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK, GCIS_PARM_DLINK_SAPI,
        sizeof( char ), &sapi);

    ces = 1;
    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK, GCIS_PARM_DLINK_CES,
        sizeof( char ), &ces);

    mode = EV_SYNC;

    ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle,
        GCIS_EXID_GETDLINKSTATE, parm_blkp, &ret_blkp, mode);

    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);
           } else {
                    printf("gc_ErrorInfo() call failed\n");
           }
    }

    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.5 Retrieve the CES and SAPI (BRI Only)

Notes: ***1.*** The GCIS_EXID_GETENDPOINT extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_GETENDPOINT extension ID is **not** supported.

2. The GCIS_EXID_GETENDPOINT extension ID applies only to BRI protocols and is **not** supported for the Dialogic® BRI/2 Board.

The GCIS_EXID_GETENDPOINT extension ID is used to retrieve the connection endpoint suffix (CES) and service access point ID (SAPI) associated with a GCEV_D_CHAN_STATUS event. The CES specifies the telephone equipment associated with the station. Currently, for BRI, eight IDs (1 – 8) are supported when used as a network-side terminal. When used as a station-side terminal, only one ID (which must have a value of 1) is supported.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the device
ext_id	GCIS_EXID_GETENDPOINT

Parameter	Input
parmblkp	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_EVENTDATAP **value** – evtdatap member of METAEVENT structure ***value_type** – void
retblkp	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI • 16 for X.25 packets over D channel **value_type** – char
mode	EV_SYNC

Code Example

```
int extGetEndPoint (LINEDEV handle, void *evtdatap)
{
    GC_PARM_DATAP parm_datap;
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_GENERIC, GCIS_PARM_EVENTDATAP,
         sizeof( void * ), evtdatap);

    mode = EV_SYNC;

    ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle, GCIS_EXID_GETENDPOINT,
           parm_blkp, &ret_blkp, mode);

    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);
           }
    }
    gc_util_delete_parm_blk( parm_blkp );
    gc_util_delete_parm_blk( ret_blkp );
    return ret_val;
}
```

4.1.6 Retrieve Frame from Application

Notes: **1.** The GCIS_EXID_GETFRAME extension ID is supported when using Dialogic® Springware Boards only. The GCIS_EXID_GETFRAME extension ID is **not** supported when using Dialogic® DM3 Boards; use the **gc_GetFrame( )** function instead.

2. The GCIS_EXID_GETFRAME extension ID is **not** supported for the Dialogic® BRI/2 Board or for the PRI DPNSS protocol.

The GCIS_EXID_GETFRAME extension ID is used to retrieve the frame received by the application. The **gc_Extension( )** function is called after a GCEV_EXTENSION event with an ext_id of GCIS_EXEV_L2FRAME is received. Each GCEV_EXTENSION event is associated with one frame. This extension function is used for the data link layer only.

Note: To enable Layer 2 access, set parameter number 24 to 01 in the firmware parameter file. When Layer 2 access is enabled, only the **gc_Extension( )** function with the **ext_id** set as either GCIS_EXID_GETFRAME or GCIS_EXID_SNDFRAME can be used (no calls can be made).

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	the board device handle of the device
ext_id	GCIS_EXID_GETFRAME
retblkp	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type**– char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided **value_type** – char array, length should not exceed MAXLEN_IEDATA
mode	EV_SYNC

Note: The **gc_Extension( )** function with **ext_id** set to GCIS_EXID_GETFRAME can be called only after a GCEV_EXTENSION(ext_id = GCIS_EXEV_L2FRAME) event is received. Refer to the protocol specific parameter file.

Code Example

```
int extGetFrame (LINEDEV handle)
{
    GC_PARM_DATAP parm_datap;
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;
    mode = EV_SYNC;

    ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle, GCIS_EXID_GETFRAME,
           parm_blkp, &ret_blkp, mode);
```

```
        if ( ret_val )
        {
                ret_val = gc_ErrorInfo(&t_Info);
                if (ret_val == GC_SUCCESS) {
                        printf("gc_ErrorInfo() successfully called\n");
                        PrintGC_INFO(&t_Info);
        }
        gc_util_delete_parm_blk( parm_blkp );
        gc_util_delete_parm_blk( ret_blkp );
        return ret_val;
}
```

4.1.7 Retrieve the Network Call Reference Value (CRV)

Notes: 1. The GCIS_EXID_GETNETCRV extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the CRV can be retrieved using **gc_GetNetCRV( )** function.

2. The GCIS_EXID_GETNETCRV extension ID is **not** supported for the Dialogic® BRI/2 Board.

The GCIS_EXID_GETNETCRV extension ID is used to retrieve the network call reference value (CRV) for a specified call reference number (CRN). The CRN is assigned during either the **gc_MakeCall( )** function for outbound calls or the **gc_WaitCall( )** function for incoming calls. If an invalid host CRN value is passed, for example, the CRN of an inactive call, the **gc_Extension( )** function will return a value <0 indicating failure.

Note: The GCIS_EXID_GETNETCRV extension ID can be used to invoke the Two B Channel Transfer (TBCT) feature. The TBCT feature is invoked by sending a FACILITY message to the network containing, among other things, the call reference values (CRVs) of the two calls to be transferred. See Section 3.1, "General ISDN Call Scenarios", on page 47 for more information on TBCT.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CRN
target_id	call reference number (CRN) of the call
ext_id	GCIS_EXID_GETNETCRV
retblkp	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_NETCRV **values** – network provided **value_type** – int
mode	EV_SYNC

Code Example

```
int UseExtGetNetCRV (CRN crn, int *netcrvp, unsigned mode)
{
    /* The GC_PARM_BLK data must point to NULL initially */
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    GC_PARM_DATAP parm_datap;
    int ret_val = 0;
```

```
    GC_INFO  t_Info;

    /* Insert the parm into the data block */
    gc_util_insert_parm_ref(&parm_blkp, GCIS_SET_GENERIC,
                            GCIS_PARM_NETCRV, sizeof(int), 0);
    ret_val = gc_Extension( GCTGT_GCLIB_CRN, crn,
                            GCIS_EXID_GETNETCRV, parm_blkp,
                            &ret_blkp, mode);
    if ( ret_val )
    {
        ret_val = gc_ErrorInfo(&t_Info);
        if (ret_val == GC_SUCCESS) {
            printf("gc_Extension() fails with GC Error 0x%xh: %s\n",
                    t_Info.gcValue, t_Info.gcMsg);
            printf("CC %d(%s) Error - 0x%xh: %s\n", t_Info.ccLibId,
                    t_Info.ccLibName, t_Info.ccValue, t_Info.ccMsg);
            printf("Additional message: %s\n", t_Info.additionalInfo);
        }
        else {
            printf("gc_ErrorInfo() call failed\n");
        }
    }
    /* Get the first parm from the data block */
    parm_datap = gc_util_next_parm(parm_blkp, NULL);

    /* Get the NetCRV from the parm data */
    memcpy(netcrvp, parm_datap->value_buf, sizeof(int));

    /* Free the Parm data block allocated when done */
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.8 Retrieve Information for a GLOBAL or NULL CRN Event

Note: The GCIS_EXID_GETNONCALLMSG extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_GETNONCALLMSG extension ID is **not** supported.

The GCIS_EXID_GETNONCALLMSG extension ID retrieves information for a GLOBAL or NULL CRN event, at the time the event occurs. The GCIS_EXID_GETNONCALLMSG extension ID must be used immediately after the event is received if the application needs the call information. The library will not queue the call information; subsequent messages on the same board device will overwrite this information if it is not retrieved immediately. NULL events correspond to messages received with a dummy, or NULL, call reference value (CRV). These messages are of significance to all calls or channels on a particular trunk, that is, they do not correspond to a particular call. Therefore, the messages are delivered on the board level device (for example, briS1). This extension ID can be used to retrieve information for the following NULL events:

- GCEV_EXTENSION with ext_id as GCIS_EXEV_INFONULL
- GCEV_EXTENSION with ext_id as GCIS_EXEV_NOTIFYNULL
- GCEV_EXTENSION with ext_id as GCIS_EXEV_FACILITYNULL

GLOBAL events correspond to messages received with a zero call reference value. These messages are of significance to all calls or channels on a particular trunk, that is, they do not correspond to a particular call. Therefore, the messages are delivered on the board level device (for example, briS1). This extension ID can be used to retrieve the information for the following GLOBAL events:

- GCEV_EXTENSION with ext_id as GCIS_EXEV_INFOGLOBAL
- GCEV_EXTENSION with ext_id as GCIS_EXEV_NOTIFYGLOBAL
- GCEV_EXTENSION with ext_id as GCIS_EXEV_FACILITYGLOBAL

Note: Some IEs may require a Call Reference Value (CRV) to be part of the contents. The Call Reference, in this case, must be the Call Reference Value assigned by the network, not the Call Reference Number (CRN) that is generated by Global Call and retrieved using the **gc_GetCRN( )** function. It is up to the application to correctly format and order the IEs. Refer to the ISDN Recommendation Q.931 or the switch specification of the application's ISDN protocol for the relevant CCITT format. See the example code for details. To receive GLOBAL and NULL events, an appropriate handler must be enabled on the board level device. See the **sr_enbhdlr( )** function in the *Dialogic® Standard Runtime Library API Programming Guide*.

The information related to a GLOBAL or NULL event must be retrieved immediately as it will be overwritten by the next event.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	board device handle of the device
ext_id	GCIS_EXID_GETNONCALLMSG
retblkp	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided **value_type** – char array, length should not exceed MAXLEN_IEDATA
mode	EV_SYNC

Code Example

```
int extGetNonCallMsg (LINEDEV handle)
{
        GC_PARM_DATAP parm_datap;
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        mode = EV_SYNC;

        ret_val = gc_Extension( GCTGT_GCLIB_CHAN, handle,
                        GCIS_EXID_GETNONCALLMSG, parm_blkp, &ret_blkp, mode);

        if ( ret_val )
        {
            ret_val = gc_ErrorInfo(&t_Info);
            if (ret_val == GC_SUCCESS) {
                   printf("gc_ErrorInfo() successfully called\n");
                   PrintGC_INFO(&t_Info);
           }
        gc_util_delete_parm_blk( parm_blkp );
        gc_util_delete_parm_blk( ret_blkp );
        return ret_val;
}
```

4.1.9 Play a User-Defined Tone

Notes: ***1.*** The GCIS_EXID_PLAYTONE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_PLAYTONE extension ID is **not** supported.

2. This extension ID is **not** supported for the Dialogic® BRI/2 Board or for PRI protocols.

The GCIS_EXID_PLAYTONE extension ID allows the application to play a user-defined tone.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle of the device
ext_id	GCIS_EXID_PLAYTONE
parmblkp	**set_id** – GCIS_SET_TONE **parm_id** – GCIS_PARM_TONE_DURATION **values** – range is 1 to 65535. Set to -1 to play forever. **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_FREQ1 **values** – range is 200 to 3100 Hz **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_AMP1 **values** – range is -40 to +3 dB **value_type** – short

Parameter	Input
	parm_id – GCIS_PARM_TONE_FREQ2 **values** – range is 200 to 3100 Hz **value_type** – unsigned short
	parm_id – GCIS_PARM_TONE_AMP2 **values** – range is -40 - +3 dB. **value_type** – short
	parm_id – GCIS_PARM_TONE_ON1 **values** – 1 to 65535 ms. Set to 1 or greater for continuous tone. **value_type** – unsigned short
	parm_id – GCIS_PARM_TONE_OFF1 **values** – range is 0 to 65534 ms. Set to 0 to play a continuous tone. **value_type** – unsigned short
mode	EV_SYNC, EV_ASYNC

Termination Events

GCEV_EXTENSION with an ext_id of GCIS_EXEV_PLAYTONE
Indicates that the tone was successfully played.

GCEV_EXTENSION with an ext_id of GCIS_EXEV_PLAYTONEFAIL
Indicates that the request to play a tone failed.

Note: The channel must be in the Idle state when calling this function. This command is a host tone command that allows the application to play a user-defined tone. This command cannot be used to set or play the firmware-applied call progress tones. The call progress tones and user-defined tones operate independently, except that when the firmware is playing a tone, the application may not play a tone on the same channel at the same time. For information on changing the firmware-applied call progress tones, see the GCIS_EXID_TONEREDEFINE extension ID description.

Code Example

```
int extPlayTone (LINEDEV handle)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        short stmp3;
        unsigned short ustmp4;
        ustmp4 = 400;
        gc_util_insert_parm_ref(&parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_DURATION, sizeof( unsigned short ), &ustmp4);
        ustmp4 = (unsigned short)350;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_FREQ1, sizeof( unsigned short ), ustmp4);

        stmp3 = (short)-10;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE, GCIS_PARM_TONE_AMP1,
                sizeof( short ), &stmp3);

        ustmp4 = (unsigned short)460;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_FREQ2, sizeof( unsigned short ), &ustmp4);
```

```
        stmp3 = (short)-10;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_AMP2, sizeof( short ), &stmp3);

        ustmp4 = (unsigned short)400;

        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_ON1, sizeof( unsigned short ), &ustmp4);

        ustmp4 = (unsigned short)0;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_OFF1, sizeof( unsigned short ), &ustmp4);

        mode = EV_SYNC

        ret_val = gc_Extension( GCTGT_GCLIB_CHAN, handle
                  GCIS_EXID_PLAYTONE, parm_blkp, &ret_blkp, mode);
        if ( ret_val )
        {
               ret_val = gc_ErrorInfo(&t_Info);
               if (ret_val == GC_SUCCESS) {
                        printf("gc_ErrorInfo() successfully called\n");
                        PrintGC_INFO(&t_Info);
               } else {
                        printf("gc_ErrorInfo() call failed\n");
               }
        }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}
```

4.1.10 Set the Logical Data Link State

The GCIS_EXID_SETDLINKSTATE extension ID is supported when using Dialogic® DM3 and Springware Boards.

The GCIS_EXID_SETDLINKSTATE extension ID asks the firmware to set the logical data link state to support specific events in your application.

Upon successful completion, the request to change the state of the logical link is accepted by the firmware. For Dialogic® DM3 Boards in asynchronous mode, a GCEV_EXTENSION event is also received. Subsequently, when the logical data link state changes, an unsolicited GCEV_D_CHAN_STATUS event is received, indicating that the state has changed.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the board device
ext_id	GCIS_EXID_SETDLINKSTATE

Parameter	Input
parmblkp	the pstruct member of **parmblkp** should point to the DLINK (Data Link Information Block) data structure followed by int. See the DLINK structure reference page in this publication, which includes a code example, for more information. **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES (Dialogic® Springware Boards only) **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI (Dialogic® Springware Boards only) **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **parm_id** – GCIS_PARM_DLINK_STATE (Dialogic® DM3 and Springware Boards) **values** – One of the following values: • DATA_LINK_DISABLED - Channel layer 2 was disabled and cannot be reestablished. The firmware attempts to release the logical link if it is currently established. The firmware does **not** allow the network side to establish the logical link if requested. • DATA_LINK_DOWN - Not supported by Dialogic® DM3 Boards. Channel layer 2 is not operational. The firmware attempts to release the logical link if it is currently established. The firmware allows the network side to establish the logical link if requested. • DATA_LINK_UP - Channel layer 2 is operational. The firmware attempts to activate the logical link if it is not already activated and allows the network side to establish the logical link if requested. **value_type** – int
mode	EV_ASYNC (Dialogic® DM3 Boards only) EV_SYNC (Dialogic® DM3 and Springware Boards)

Notes:

1. There needs to be a sufficient amount of time between bringing down the data link layer and bringing it up. This is necessary to allow time for the network side to release its resources and declare the data link down before the network side tries to reestablish the connection.
2. Although GCIS_EXID_SETDLINKSTATE can be used for PRI, it is somewhat limited in scope. In PRI, after Layer 2 is brought down (DATA_LINK_DOWN state), the firmware will try to reestablish the link after the timer expires.
3. For Dialogic® DM3 Boards, if the **gc_Extension( )** function is called before the previous transaction finished, the function will terminate with an EGC_ILLSTATE error that corresponds to “Invalid state”.

Code Example

Note: The following example applies when using Dialogic® Springware Boards only.

```
int extSetDLinkState (LINEDEV handle)
```

```
{
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO  t_Info;
    char sapi, ces;
    int state;

    sapi = 0;
    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK, GCIS_PARM_DLINK_SAPI,
                             sizeof( char ), &sapi);
    ces = 1;
    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK, GCIS_PARM_DLINK_CES,
                             sizeof( char ), &ces);
    state = DATA_LINK_UP;
    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK,
                            GCIS_PARM_DLINK_STATE, sizeof( int ), &state);

    mode = EV_SYNC;

    ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle,
                  GCIS_EXID_SETDLINKSTATE, parm_blkp, &ret_blkp, mode);
    if ( ret_val )
    {
           ret_val = gc_ErrorInfo(&t_Info);
           if (ret_val == GC_SUCCESS) {
                    printf("gc_ErrorInfo() successfully called\n");
                    PrintGC_INFO(&t_Info);
           } else {
                    printf("gc_ErrorInfo() call failed\n");
           }
    }

    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );
    return ret_val;
}
```

4.1.11 Send Frame to the Data Link Layer

Notes: ***1.*** The GCIS_EXID_SNDFRAME extension ID is supported when using Dialogic® Springware Boards only. The GCIS_EXID_SNDFRAME extension ID is **not** supported when using Dialogic® DM3 Boards; use the **gc_SndFrame( )** function instead.

2. This extension ID is **not** supported for the Dialogic® BRI/2 Board.

The GCIS_EXID_SNDFRAME extension ID is used to send a frame to the data link layer. When the data link layer is successfully established, the application will receive a GCEV_D_CHAN_STATUS event. If the data link layer is not established before the function is called, the function will be returned with a value <0 indicating function failure.

Note: To enable Layer 2 access, set parameter number 24 to 01 in the firmware parameter file. When Layer 2 access is enabled, only the **gc_Extension( )** function with the **ext_id** parameter set to GCIS_EXID_GETFRAME can be used (no calls can be made).

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CRN
target_id	call reference number (CRN) of the call
ext_id	GCIS_EXID_SNDFRAME
parmblkp	the pstruct member of **parmblkp** should point to the L2_BLK data structure. See the L2_BLK structure reference page in this publication, which includes a code example. **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided **value_type** – char array, length should not exceed MAXLEN_IEDATA
mode	EV_SYNC

Note: The data link layer must be successfully established before the **gc_Extension( )** function with **ext_id** GCIS_EXID_SndFrame is called.

Code Example

```
int extSndFrame (LINEDEV handle)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        int indicator
        char sapi, ces, ie_data[255];

        sapi = 0;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK,
              GCIS_PARM_DLINK_SAPI, sizeof( char ), &sapi);

        ces = 1;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK,
              GCIS_PARM_DLINK_CES, sizeof( char ), &ces);

        InitSndFrameBlk(ie_data);
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_IE, GCIS_PARM_IEDATA,
               13, ie_data);

        mode = EV_SYNC;
```

```
        ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle, GCIS_EXID_SNDFRAME,
               parm_blkp, &ret_blkp, mode);
        if ( ret_val )
        {
               ret_val = gc_ErrorInfo(&t_Info);
               if (ret_val == GC_SUCCESS) {
                       printf("gc_ErrorInfo() successfully called\n");
                       PrintGC_INFO(&t_Info);
               } else {
                   printf("gc_ErrorInfo() call failed\n");
               }
        }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}

void InitSndFrameBlk (char *data)
{
        data [0] = 0x08;     /* Protocol discriminator */
        data [1] = 0x02;     /* CRN length - 2 bytes */
        data [2] = 0x03;     /* CRN = 8003 */
        data [3] = 0x80;
        data [4] = 0x6e;     /* msg type = NOTIFY */

        /* The first IE */
        data [5] = 0x27;     /* IE type = 27 (NOTIFY) */
        data [6] = 0x01;     /* The length of NOTIFY */
        data [7] = 0xF1;     /* Notify indication */

        /* The second IE */
        data [8] = 0x76;     /* IE type = 76 (REDIRECTION) */
        data [9] = 0x03;     /* length of redirection */
        data [10] = 0x01;    /* unknown type and E164 plan */
        data [11] = 0x03;    /* network provides presentation */
        data [12] = 0x8D;    /* reason = transfer */
        }
```

4.1.12 Send a Non-Call State Related ISDN Message

Note: The GCIS_EXID_SNDMSG extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, a non-call state related ISDN message can be sent using the **gc_SndMsg( )** function. See Section 8.2.38, "gc_SndMsg( ) Variances for ISDN", on page 245 for more information.

The GCIS_EXID_SNDMSG extension ID is used to send a non-call state related ISDN message to the network over the D channel, while a call exists. The data is sent transparently over the D channel data link using the LAPD (Layer 2) protocol.

Note: The message must be sent over a channel that has a call reference number assigned to it.

For BRI, this extension function is used to invoke supplemental services, such as Called/Calling Party Identification, Call Transfer, and Message Waiting. The services are invoked by sending Facility Messages or Notify Messages to the switch. Upon receipt of the message, the network may return a NOTIFY message to the user. The NOTIFY message can be retrieved by calling the **gc_GetCallInfo( )** function. For more information on invoking supplemental services, see Section 12.3, "BRI Supplemental Services", on page 308.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CRN
target_id	call reference number (CRN) of the call
ext_id	GCIS_EXID_SNDMSG
parmblkp	pstruct member of parmblkp should point to the memory block containing integer (msg_type) followed by the IE_BLK data structure. See also the IE_BLK structure reference page in this publication. **set_id** – GCIS_SET_SNDMSG **parm_id** – GCIS_PARM_SNDMSGTYPE **values** – For all protocols, one of the following: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus For DPNSS only, one of the following: • SndMsg_Divert • SndMsg_Intrude • SndMsg_NSI • SndMsg_Transfer • SndMsg_Transit For BRI 5ESS only, one of the following: • SndMsg_Drop • SndMsg_DropAck • SndMsg_DropRej • SndMsg_Redirect **value_type** – int **set_id** – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided **value_type** – char array, length should not exceed MAXLEN_IEDATA
mode	EV_SYNC

Descriptions of the message types for DPNSS are provided in Section 12.2, “DPNSS IEs and Message Types”, on page 301.

Code Example

```
int extSndMsg (CRN crn)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        unsigned char ie_data[255];
        int msg;
        unsigned char length;

        msg = SndMsg_Notify;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SNDMSG,
                GCIS_PARM_SNDMSGTYPE, sizeof( int ), &msg);

        InitSndMsgBlk (ie_data, msg, &length);
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_IE, GCIS_PARM_IEDATA,
                length, ie_data);

        mode = EV_SYNC;

        ret_val = gc_Extension( GCTGT_GCLIB_CRN, crn,
               GCIS_EXID_SNDMSG, parm_blkp, &ret_blkp, mode);
        if ( ret_val )
        {
               ret_val = gc_ErrorInfo(&t_Info);
               if (ret_val == GC_SUCCESS) {
                        printf("gc_ErrorInfo() successfully called\n");
                        PrintGC_INFO(&t_Info);

               } else {
                        printf("gc_ErrorInfo() call failed\n");
               }
        }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}

static void InitSndMsgBlk (unsigned char *ie_blk_ptr, int msgtype, unsigned char *lenp)
{
         switch (msgtype)
         {
         case SndMsg_Notify:              /* Notify */
             *lenp = 3;
             ie_blk_ptr[0] = 0x27;   /* Notify Indicator IE (0x27) */
             ie_blk_ptr[1] = 0x01;   /* IE Length */

             ie_blk_ptr[2] = 0x81;   /* user resumed */
             break;

         case SndMsg_Status:
             *lenp  = 0x07;

             ie_blk_ptr[0] = 0x08; /*cause IE*/
             ie_blk_ptr[1] = 0x02; /*length*/
             ie_blk_ptr[2] = 0x80; /**/
             ie_blk_ptr[3] = 0x1F; /*cause value*/

             ie_blk_ptr[4] = 0x14; /*call state IE*/
             ie_blk_ptr[5] = 0x01; /*length*/
             ie_blk_ptr[6] = 0x0A; /*call state*/

             break;
```

```
        default:
            break;
        }
        return;
}
```

4.1.13 Send a Non-Call Related ISDN Message

Note: The GCIS_EXID_SNDNONCALLMSG extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_SNDNONCALLMSG extension ID is **not** supported.

The GCIS_EXID_SNDNONCALLMSG extension ID is used to send a non-call related ISDN message to the network over the D channel. This extension ID specifies the ISDN CRN Type as either:

GLOBAL CRN
: pertaining to all calls or channels on a trunk

NULL CRN
: not related to any particular call

Unlike the GCIS_EXID_SNDMSG extension ID, this extension ID does not require a call reference number (CRN) to transmit the outgoing message.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	board device handle of the device
ext_id	GCIS_EXID_SNDNONCALLMSG
parmblkp	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_CRNTYPE **values** – One of the following: • GLOBAL CRN • NULL CRN **value_type** – int **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – One of the following: • 0 for PRI • 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_SNDMSG **parm_id** – GCIS_PARM_SNDMSGTYPE **values** – For all protocols, one of the following: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus For DPNSS only, one of the following: • SndMsg_Divert • SndMsg_Intrude • SndMsg_NSI • SndMsg_Transfer • SndMsg_Transit For Custom BRI 5ESS only, one of the following: • SndMsg_Drop • SndMsg_DropAck • SndMsg_DropRej • SndMsg_Redirect **value_type** – int

Parameter	Input
	set_id – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided **value_type** – char array, length should not exceed MAXLEN_IEDATA
mode	EV_SYNC

Note: Some IEs may require a Call Reference Value (CRV) to be part of the contents. The call reference in this case, must be the Call Reference Value assigned by the network, not the Call Reference Number (CRN) that is assigned by Global Call and retrieved using the **gc_GetCRN( )** function. It is up to the application to correctly format and order the IEs. Refer to the ISDN Recommendation Q.931 or the switch specification of the application's ISDN protocol for the relevant CCITT format.

Code Example

```
int extSndNonCallMsg (LINEDEV handle)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        int indicator
        char sapi, ces, ie_data[255];
        int msg;
        unsigned char length;

        gc_util_insert_parm_val( &parm_blkp, GCIS_SET_GENERIC,
             GCIS_PARM_CRNTYPE, sizeof( int ), GLOBAL_CRN);

        sapi = 0;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK,
            GCIS_PARM_DLINK_SAPI, sizeof( char ), &sapi);

        ces = 1;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_DLINK,
             GCIS_PARM_DLINK_CES, sizeof( char ), &ces);

        msg = SndMsg_Notify;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SNDMSG,
             GCIS_PARM_SNDMSGTYPE, sizeof( int ), &msg);

//See previous section on Send a Non-Call State Related ISDN Message
        InitSndMsgBlk (ie_data, msg, &length);
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_IE,
             GCIS_PARM_IEDATA, length, ie_data);

        mode = EV_SYNC;

        ret_val = gc_Extension(GCTGT_GCLIB_CHAN, handle,
             GCIS_EXID_SNDNONCALLMSG,
        parm_blkp, &ret_blkp, mode);
        if ( ret_val )
        {
               ret_val = gc_ErrorInfo(&t_Info);
               if (ret_val == GC_SUCCESS) {
                        printf("gc_ErrorInfo() successfully called\n");
```

```
                PrintGC_INFO(&t_Info);
            } else {
                printf("gc_ErrorInfo() call failed\n");
            }
        }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}
```

4.1.14 Stop Currently Playing Tone (BRI Only)

Notes: ***1.*** The GCIS_EXID_STOPTONE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_STOPTONE extension ID is **not** supported.

2. This function is **not** supported for the Dialogic® BRI/2 Board or PRI protocols.

The GCIS_EXID_STOPTONE extension ID forces the termination of a tone that is currently playing on a channel. The function forces a channel that is in the playing state to become idle. Running this function asynchronously initiates the function without affecting processes on other channels. Running this function synchronously within a process does not block other processing, allowing other processes to continue to be serviced.

This extension ID allows the application to stop the playing of user-defined tones only. This command cannot be used to stop the playing of the firmware-applied call progress tones. The firmware-applied call progress tones and user-defined tones operate independently, except that when the firmware is playing a call progress tone, the application may not play a user-defined tone on the same channel at the same time.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle of the device
ext_id	GCIS_EXID_STOPTONE
mode	EV_SYNC, EV_ASYNC

Termination Events

GCIS_EXEV_STOPTONE
: Indicates that the tone was successfully stopped and the channel was returned to the idle state.

GCIS_EXEV_STOPTONEFAIL
: Indicates that the request to stop a tone and return the channel to the idle state failed.

Notes: ***1.*** If an I/O function terminates due to another reason before the **gc_Extension( )** function with the GCIS_EXTID_STOPTONE extension ID is issued, the reason for termination will not indicate that **gc_Extension( )** with GCIS_EXID_STOPTONE was called.

2. In asynchronous mode, if the application tries to stop a tone that is already stopped, you will receive the GCEV_EXTENSION (ext_id = GCIS_EXEV_STOPTONEFAIL) termination event. Using the **gc_ResultMsg( )** function will retrieve the error code ERR_TONESTOP.

3. In synchronous mode, if the application tries to stop a tone that is already stopped, the function will fail. Using the **gc_ResultMsg( )** function will retrieve the error code ERR_TONESTOP.

4. When calling the **gc_Extension( )** function with the GCIS_EXID_STOPTONE extension ID from a signal handler, the mode parameter must be set to EV_ASYNC.

Code Example

```
int extStopTone (LINEDEV handle)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        GC_INFO  t_Info;
        int indicator;
        int ret_val = 0;

        ret_val = gc_Extension( GCTGT_GCLIB_CHAN, handle
                   GCIS_EXID_STOPTONE, parm_blkp, &ret_blkp, mode);
        if ( ret_val )
    {
        ret_val = gc_ErrorInfo(&t_Info);
        if (ret_val == GC_SUCCESS) {
               printf("gc_ErrorInfo() successfully called\n");
               PrintGC_INFO(&t_Info);
       } else {
               printf("gc_ErrorInfo() call failed\n");
       }
    }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}
```

4.1.15 Redefine Call Progress Tone Attributes (BRI Only)

Notes: ***1.*** The GCIS_EXID_TONEREDEFINE extension ID is supported when using Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, the GCIS_EXID_TONEREDEFINE extension ID is **not** supported.

2. This function is **not** supported for the Dialogic® BRI/2 Board or for PRI protocols.

The GCIS_EXID_TONEREDEFINE extension ID redefines a call progress tone's attributes in the tone template table. The tone template table resides in the firmware and is used during call establishment. The template contains common call progress tone types and is preset to default values at initialization. The current template has a total of eight entries, of which only four are defined. The other four are reserved for future use.

The **gc_Extension( )** function with the GCIS_EXID_TONEREDEFINE extension ID allows you to redefine the existing tone template, but not the functional meanings of the call progress tones.

Parameter Inputs

The following table provides the parameter inputs for the **gc_Extension( )** function.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle of the device
ext_id	GCIS_EXID_TONEREDEFINE
parmblkp	**set_id** – GCIS_SET_CALLPROGRESS **parm_id** – GCIS_PARM_CALLPROGRESSTONE_TYPE **values** – One of the following: • 0x01: Dialtone • 0x02: Busytone • 0x03: Reorder • 0x04: Ringback **value_type** – unsigned char **set_id** – GCIS_SET_TONE **parm_id** – GCIS_PARM_TONE_DURATION **values** – range is 1 to 65535. Set to -1 to play forever. **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_FREQ1 **values** – range is 200 to 3100 Hz **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_AMP1 **values** – range is -40 to +3 dB **value_type** – short **parm_id** – GCIS_PARM_TONE_FREQ2 **values** – range is 200 to 3100 Hz **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_AMP2 **values** – range is -40 - +3 dB **value_type** – short **parm_id** – GCIS_PARM_TONE_ON1 **values** – 1 to 65535 ms. Set to 1 or greater for continuous tone. **value_type** – unsigned short **parm_id** – GCIS_PARM_TONE_OFF1 **values** – range is 0 to 65534 ms. Set to 0 to play a continuous tone. **value_type** – unsigned short
mode	EV_SYNC or EV_ASYNC

Termination Events

CCEV_EXEV_TONEREDEFINE
Indicates that the tone was successfully redefined.

CCEV_EXEV_TONEREDEFINEFAIL
Indicates that the function failed.

Code Example

```
int extTONEREDEFINE(LINEDEV handle)
{
        GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
        unsigned long mode;
        int ret_val = 0;
        GC_INFO  t_Info;
        short stmp3;
        unsigned short ustmp4;
        ustmp4 = 400;
        gc_util_insert_parm_ref(&parm_blkp, GCIS_SET_TONE,
                 GCIS_PARM_TONE_DURATION, sizeof( unsigned short ), &ustmp4);
        ustmp4 = (unsigned short)350;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_FREQ1, sizeof( unsigned short ), ustmp4);

        stmp3 = (short)-10;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE, GCIS_PARM_TONE_AMP1,
                sizeof( short ), &stmp3);

        ustmp4 = (unsigned short)460;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_FREQ2, sizeof( unsigned short ), &ustmp4);

        stmp3 = (short)-10;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_AMP2, sizeof( short ), &stmp3);

        ustmp4 = (unsigned short)400;

        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_ON1, sizeof( unsigned short ), &ustmp4);

        ustmp4 = (unsigned short)0;
        gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_TONE,
                GCIS_PARM_TONE_OFF1, sizeof( unsigned short ), &ustmp4);

        mode = EV_SYNC;

        ret_val = gc_Extension( GCTGT_GCLIB_CHAN, handle
                  GCIS_EXID_TONEREDEFINE, parm_blkp, &ret_blkp, mode);
        if ( ret_val )
        {
               ret_val = gc_ErrorInfo(&t_Info);
               if (ret_val == GC_SUCCESS) {
                        printf("gc_ErrorInfo() successfully called\n");
                        PrintGC_INFO(&t_Info);
               } else {
                        printf("gc_ErrorInfo() call failed\n");
               }
        }

        gc_util_delete_parm_blk( ret_blkp );
        gc_util_delete_parm_blk( parm_blkp );
        return ret_val;
}
```

4.2 Operations Performed Using RTCM

The following sections describe the ISDN-specific operations that can be performed using the Dialogic® Global Call API Real Time Configuration Management (RTCM) capability, more specifically, the **gc_GetConfigData( )**, **gc_SetConfigData( )**, and **gc_QueryConfigData( )** functions. A brief summary of RTCM, the various operations that can be performed using this capability, and an example showing the use or the **gc_SetConfigData( )** fucntion are described in the following topics:

4.2.1 RTCM Summary

There are three Global Call RTCM functions:

gc_GetConfigData( )
: used to obtain configuration parameter data for a given target object

gc_SetConfigData( )
: used to update configuration parameter data for a given target object

gc_QueryConfigData( )
: used to obtain other related data based on the source data from a target object

Target objects are identified by the **target_type** parameter, which consists of the type of physical entity (for example, a board device) and the software module that controls it (for example, cclib), and the **target_id** parameter, which is the identifier of the specific target object (for example, a line device ID). The **target_datap** parameter specifies the pointer to the GC_PARM_BLK structure. The structure contains the parameter configuration data to be retrieved or updated. It is the Global Call application's responsibility to allocate an appropriate-size data block memory for the configuration parameters (GC_PARM_BLK) and to insert parameter information (such as the set ID, parm ID, value buffer size, value buffer, and value data) into the GC_PARM_BLK data block.

The sections that follow provide a list of ISDN parameters that can be retrieved or updated using the RCTM functions. The tables list all the parameters and type of value_buf in the target_datap (of type GC_PARM_BLK) argument of the **gc_GetConfigData( )** and **gc_SetConfigData( )** functions. The parameter set IDs and parameter IDs are described in Chapter 9, "ISDN-Specific Parameter Reference".

4.2.2 Set/Retrieve Configuration of a Logical Link (BRI Only)

Note: This functionality is supported when using Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF
target_id	board device handle
GC_PARM_BLK	**set_id** – GCIS_SET_DLINKCFG **parm_id** – GCIS_PARM_DLINKCFG_TEI **values** – One of the following: • 0 to 63 for manual TEIs (chosen by the user side) • AUTO_TEI for automatic TEIs (chosen by the network side) **value_type** – char **parm_id** – GCIS_PARM_DLINKCFG_STATE **values** – One of the following: • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED **value_type** – int **parm_id** – GCIS_PARM_DLINKCFG_PROTOCOL **values** – One of the following: • DATA_LINK_PROTOCOL_Q931 • DATA_LINK_PROTOCOL_X25 **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.3 Set Configuration of Digital Subscriber Loop (BRI Only)

Note: This functionality is supported when using Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

The appropriate **gc_SetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF
target_id	board device handle
GC_PARM_BLK	**set_id** – GCIS_SET_DCHANCFG **parm_id** – GCIS_PARM_DCHANCFG_L2ACCESS **values** – One of the following: • LAYER_2_ONLY: ISDN access at layer 2. If this is selected then no other parameters are required. • FULL_ISDN_STACK: ISDN access at L3 call control. **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_SWITCHTYPE **values** – One of the following: • ISDN_BRI_5ESS = ATT 5ESS BRI • ISDN_BRI_DMS100 = Northern Telecom DMS100 BRI • ISDN_BRI_NTT = Japanese INS-Net 64 BRI • ISDN_BRI_NET3 = EuroISDN BRI • ISDN_BRI_NI1 = National ISDN 1 • ISDN_BRI_NI2 = National ISDN 2 **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_SWITCHSIDE **values** – One of the following: • USER_SIDE = User side of ISDN protocol • NETWORK_SIDE = Network side of ISDN protocol **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_NUMENDPOINTS **values** – 1 to MAX_DLINK range, where MAX_DLINK is currently set to 8 **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_FIRMWARE_FEATUREMASKA **values** – One of the following: • NO_PCM_TONE • ULAW_PCM_TONE • ALAW_PCM_TONE • DEFAULT_PCM_TONE • SENDING_COMPLETE_ATTACH • USER_PERST_L2_ACT • HOST_CONTROLLED_RELEASE **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_TEIASSIGNMENT **values** – One of the following: • AUTO_TEI_TERMINAL = auto TEI assigning Term • FIXED_TEI_TERMINAL = Fixed TEI assigning Term **value_type** – char

Parameter	Input
	parm_id – GCIS_PARM_DCHANCFG_FIXEDTEIVALUE **values** – in the range 0 to 63 **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_AUTOINITFLAG **values** – One of the following: • AUTO_INIT_TERMINAL • NON_INIT_TERMINAL **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_SPID **values** – ASCII digit string limited to the digits 0 to 9 and limited in length to MAX_SPID_SIZE **value_type** – string **parm_id** – One of the following: • GCIS_PARM_DCHANCFG_TMR302 • GCIS_PARM_DCHANCFG_TMR303 • GCIS_PARM_DCHANCFG_TMR304 • GCIS_PARM_DCHANCFG_TMR305 • GCIS_PARM_DCHANCFG_TMR306 • GCIS_PARM_DCHANCFG_TMR308 • GCIS_PARM_DCHANCFG_TMR309 • GCIS_PARM_DCHANCFG_TMR310 • GCIS_PARM_DCHANCFG_TMR312 • GCIS_PARM_DCHANCFG_TMR313 • GCIS_PARM_DCHANCFG_TMR318 • GCIS_PARM_DCHANCFG_TMR319 • GCIS_PARM_DCHANCFG_TMR322 **values** – See Q.931 specification and corresponding switch specifications for exact definitions and default values for these timers. Not all timers are applicable to all of the switches. Specified values are in 10 millisecond increments. For example, a specified value of 100 is equivalent to 1 second. Possible values are: • 0 = Default value for switch • -1 = Default value for switch **value_type** – long
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.4 Set/Retrieve Bearer Channel Information Transfer Capability

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, bearer channel information can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCSET_CHAN_CAPABILITY **parm_id** – GCPARM_TYPE **values** – One of the following: • GCCAPTYPE_AUDIO • GCCAPTYPE_UNDEFINED • GCCAPTYPE_3KHZ_AUDIO • GCCAPTYPE_7KHZ_AUDIO • GCCAPTYPE_VIDEO **value_type** – unsigned char
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.5 Set/Retrieve Bearer Channel Information Transfer Mode

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, bearer channel information transfer mode cannot be set, but it can be retrieved using **gc_GetSigInfo**( ).

The appropriate **gc_SetConfigData**( ) and **gc_GetConfigData**( ) function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_BEARERCHNL **parm_id** – GCIS_PARM_TRANSFERMODE **values** – One of the following: • ISDN_ITM_CIRCUIT • ISDN_ITM_PACKET **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.6 Set/Retrieve Bearer Channel Information Transfer Rate

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, bearer channel information transfer rate can be set using **gc_MakeCall**( ) and retrieved using **gc_GetSigInfo**( ).

The appropriate **gc_SetConfigData**( ) and **gc_GetConfigData**( ) function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_BEARERCHNL **parm_id** – GCIS_PARM_TRANSFERRATE **values** – One of the following: • BEAR_RATE_64KBPS • BEAR_RATE_128KBPS • BEAR_RATE_384KBPS • BEAR_RATE_1536KBPS • BEAR_RATE_1920KBPS **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.7 Set/Retrieve Layer 1 Protocol to Use on Bearer Channel

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, layer 1 protocol (for bearer channel use) can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCSET_CHAN_CAPABILITY **parm_id** – GCPARM_CAPABILITY **values** – One of the following: • GCCAP_DATA_CCITTV110 • GCCAP_AUDIO_g711Ulaw64k • GCCAP_AUDIO_g711Ulaw56k • GCCAP_AUDIO_g711Alaw64k • GCCAP_AUDIO_g711Alaw56k • GCCAP_AUDIO_G721ADPCM • GCCAP_VIDEO_h261 • GCCAP_DATA_CCITTV120 • GCCAP_DATA_CCITTX31 • GCCAP_DATA_nonStandard **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.8 Set/Retrieve Logical Data Link State

Note: This functionality is supported for Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the board device
parmblkp	the pstruct member of **parmblkp** should point to the DLINK (Data Link Information Block) data structure followed by int. See the DLINK structure reference page in this publication, which includes a code example, for more information. **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – 1-8 for BRI when used as a network-side terminal **value_type** - char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **parm_id** – GCIS_PARM_DLINKCFG_STATE **values** – One of the following: • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED **value_type** – int
retblkp	pointer to the buffer containing the requested data link state value **set_id** – GCIS_SET_DLINK **parm_id** - GCIS_PARM_DLINKCFG_STATE **values** – One of the following: • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED **value_type** - int
mode	EV_ASYNC, EV_SYNC

4.2.9 Set/Retrieve User Rate to Use on Bearer Channel (Layer 1 Rate)

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, user rate (for bearer channel use) can be set using **gc_SetInfoElem( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCSET_CHAN_CAPABILITY **parm_id** – GCPARM_RATE **values** – One of the following: • GCCAPRATE_EINI460 • GCCAPRATE_600 • GCCAPRATE_1200 • GCCAPRATE_2400 • GCCAPRATE_3600 • GCCAPRATE_4800 • GCCAPRATE_7200 • GCCAPRATE_8000 • GCCAPRATE_9600 • GCCAPRATE_14400 • GCCAPRATE_16000 • GCCAPRATE_19200 • GCCAPRATE_32000 • GCCAPRATE_48000 • GCCAPRATE_56000 • GCCAPRATE_64000 • GCCAPRATE_134 • GCCAPRATE_100 • GCCAPRATE_75_1200 • GCCAPRATE_1200_75 • GCCAPRATE_50 • GCCAPRATE_75 • GCCAPRATE_110 • GCCAPRATE_150 • GCCAPRATE_200 • GCCAPRATE_300 • GCCAPRATE_12000 • GCCAPRATE_DEFAULT **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.10 Set/Retrieve Called Number Type

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, called number type can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_ADDRESS **parm_id** – GCIS_PARM_CALLEDADDRESSTYPE **values** – One of the following: • GCADDRTYPE_INTL – international number for international call. (Verify availability with service provider.) • GCADDRTYPE_NAT – national number for call within national numbering plan (accepted by most networks) • GCADDRTYPE_LOC – subscriber number for a local call. (Verify availability with service provider.) **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.11 Set/Retrieve Called Number Plan

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, called number plan can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_ADDRESS **parm_id** – GCIS_PARM_CALLEDADDRESSPLAN **values** – One of the following: • GCADDRPLAN_UNKNOWN • GCADDRPLAN_ISDN • GCADDRPLAN_TELEPHONY • GCADDRPLAN_PRIVATE **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.12 Set/Retrieve Calling Number Type

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, calling number type can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_ADDRESS **parm_id** – GCIS_PARM_CALLINGADDRESSTYPE **values** – One of the following: • GCADDRTYPE_INTL • GCADDRTYPE_NAT • GCADDRTYPE_LOC **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.13 Set/Retrieve Calling Number Plan

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, calling number plan can be set using **gc_MakeCall( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_ADDRESS **parm_id** – GCIS_PARM_CALLINGADDRESSPLAN **values** – One of the following: • GCADDRPLAN_UNKNOWN • GCADDRPLAN_ISDN • GCADDRPLAN_TELEPHONY **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.14 Set/Retrieve Calling Presentation Indicator

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, calling presentation indicator can be set using **gc_SetInfoElem( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_CALLINGPRESENTATION **values** – PRESENTATION_ALLOWED **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.15 Set/Retrieve Calling Screening Indicator

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, calling screening indicator can be set using **gc_SetInfoElem( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_CALLINGSCREENING **values** – user-provided **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.16 Set/Retrieve Multiple IE Buffer Size

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, multiple IE buffer size cannot be retrieved, but it can be set using **gc_SetParm( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_RECEIVEINFOBUF **values** – range of 1 to MAX_RECEIVE_INFO_BUF **value_type** – int

Parameter	Input
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.17 Set SPID number on BRI (North America only)

Note: This functionality is supported for Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

The appropriate **gc_SetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF
target_id	board device handle
GC_PARM_BLK	**set_id** – GCIS_SET_DCHANCFG **parm_id** – GCIS_PARM_DCHANCFG_SPID **values** – ASCII digit string limited to the digits 0-9 and limited in length to MAX_SPID_SIZE **value_type** – char
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.18 Set/Retrieve Subaddress Number on BRI (User-Side Switch Only)

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, subaddress number can be set using **gc_SetInfoElem( )** and retrieved using **gc_GetSigInfo( )**.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_SUBADDRESSNUMBER **values** – unsigned char array of max length 255 **value_type** – unsigned char
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.19 Set/Retrieve Directory Number on BRI (User-Side Switch Only)

Note: This functionality is supported for Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

The appropriate **gc_SetConfigData( )** and **gc_GetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_GENERIC **parm_id** – GCIS_PARM_DIRECTORYNUMBER **values** – unsigned char array of max length 255 **value_type** – unsigned char
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.20 Set ISDN-Specific Event Masks

Note: This functionality is supported for Dialogic® Springware Boards only. When using Dialogic® DM3 Boards, ISDN-specific masks can be set using **gc_SetEvtMsk( )**. See Section 8.2.33, “gc_SetEvtMsk( ) Variances for ISDN”, on page 239 for more information.

The appropriate **gc_SetConfigData( )** function parameter values are shown in the following table.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)
GC_PARM_BLK	**set_id** – GCIS_SET_EVENTMSK **parm_id** – At least one of the following should be present and applies only to the **gc_SetConfigData( )** function: • GCIS_PARM_ADDMSK • GCIS_PARM_SUBMSK • GCIS_PARM_SETMSK **values** – One of the following: • GCISMSK_STATUS † • GCISMSK_STATUS_ENQUIRY † • GCISMSK_TMREXPEVENT † • GCMSK_ALERTING • GCMSK_PROC_SEND • GCMSK_PROCEEDING • GCMSK_PROGRESS • GCMSK_SETUP_ACK **Note:** † indicates masks that are supported on PRI only. **value_type** – int

Parameter	Input
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

4.2.21 Example of gc_SetConfigData( )

The following sample code provides examples of using the **gc_SetConfigData( )** function to update and obtain ISDN parameter data.

Note: The following code applies when using Dialogic® Springware Boards only. The **gc_SetConfigData( )** function is **not** supported when using Dialogic® DM3 Boards.

```
int SetConfigDataChan(LINEDEV linedev)

{
     int retcode;
     long request_id;
     GC_PARM_BLKP target_datap=NULL;
     PARM_INFO parm_info;
     int         gc_error;             /* Global Call Error */
     int         cclibid;              /* CC Library ID */
     long        cc_error;             /* Call Control Library error code */
     char        *msg;                 /* pointer to error message string */

     strcpy(parm_info.parmdata, "12345678987");
     parm_info.parmdatalen = strlen(parm_info.parmdata);

     gc_util_insert_parm_val(&target_datap, GCIS_ADD_EVENTMSK,
          GCIS_PARM_SETMSK, (unsigned char)sizeof(int), GCISMSK_STATUS_ENQUIRY);
     gc_util_insert_parm_val(&target_datap, GCIS_ADD_EVENTMSK,
          GCIS_PARM_SETMSK, (unsigned char)sizeof(int), GCISMSK_STATUS);
     gc_util_insert_parm_val(&target_datap, GCIS_ADD_EVENTMSK,
          GCIS_PARM_SETMSK, (unsigned char)sizeof(int), GCISMSK_TMREXPEVENT);

     gc_util_insert_parm_val(&target_datap, GCSET_CALL_CONFIG,
          GCPARM_MIN_INFO, (unsigned char)sizeof(int), 5);

     retcode=gc_SetConfigData(GCTGT_CCLIB_CHAN, linedev target_datap, 60,
                 GCUPDATE_IMMEDIATE,&request_id, EV_SYNC);
     printf("gc_SetConfigData(GCTGT_CCLIB_CHAN, 0x%X, target_datap, 60",
                 linedev);
     gc_util_delete_parm_blk(target_datap);
     if (retcode==-1)
    {
          gc_ErrorValue( &gc_error, &cclibid, &cc_error);
          gc_ResultMsg( LIBID_GC, (long) gc_error, &msg);
          gc_ResultMsg( cclibid, cc_error, &msg);
     }

     return retcode;
}
```

4.3 Responding to a Service Request (BRI Only)

Note: The following information applies when using Dialogic® Springware Boards only. Dialogic® DM3 Boards do not support the service request feature.

The Dialogic® Global Call Service Request (GCSR) capability provides support for service requests. See the *Dialogic® Global Call API Programming Guide* for more information. The level of support provided for ISDN BRI is described in the following topics:

- Overview of Service Request Support
- Using gc_RespService( )
- Supported Service Request Events

4.3.1 Overview of Service Request Support

In BRI North American terminal initialization, the terminal equipment (TE) registration request goes to the network side. The firmware sends the information on its own. The application, when used as the Network side, receives a GCEV_SERVICEREQ event as notification of a TE registration request. On receiving this event, the application evaluates the Service Profile Interface ID (SPID) received and either rejects or accepts the registration request. The application then conveys its result to the network using the **gc_RespService( )** function to send a GCEV_SERVICERESP event to indicate whether the request is accepted or rejected. If the request is accepted, the terminal is then fully initialized.

Note: The **gc_RespService( )** function can be called on a board device handle only.

Global Call also defines the **gc_ReqService( )** function, which is not used for ISDN protocols. The device registration is automatically generated when the device is initialized, so the Service Request feature is essentially used in a response-only manner by the network side.

The following sections describe the **gc_RespService( )** function as it relates to ISDN and the corresponding events. The set and parameter IDs are described in Chapter 9, "ISDN-Specific Parameter Reference".

4.3.2 Using gc_RespService( )

Notes: ***1.*** This **gc_RespService( )** function is supported for Dialogic® Springware Boards only; not supported when using Dialogic® DM3 Boards.

2. This function is not supported for the Dialogic® BRI/2 Board.

3. This function applies only to BRI North American terminal protocols used as the network side.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF
target_id	board device handle
datap	**set_id** – GCSET_SERVREQ **parm_id** – PARM_SERVICEID **value** – 0 **value_type** – int **parm_id** – PARM_REQTYPE **value** – 0 **value_type** – int

Parameter	Input
	parm_id – PARM_ACK **values** – Any of the Q.931 cause values **value_type** – int **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_SERVREQ **parm_id** – GCIS_PARM_SERVREQ_CAUSEVALUE **values** – One of the following: • NETWORK_OUT_OF_ORDER • BAD_INFO_ELEM • INVALID_ELEM_CONTENTS • TIMER_EXPIRY • PROTOCOL_ERROR **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_USID **values** – range is 01 – FF. 00 signifies default. **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_TID **values** – range is 01 – FF. 00 signifies default. **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_INTERPRETER – Specifies how the usid and tid values are to be interpreted. **values** – One of the following: • 0 • 1 **value_type** – unsigned char
mode	EV_SYNC

Code Example

```
int extRespService (LINEDEV handle)
{
    GC_PARM_BLKP parm_blkp = NULL, ret_blkp = NULL;
    unsigned long mode;
    int ret_val = 0;
    GC_INFO t_Info;

    short stmp3;
    unsigned short ustmp4;

    gc_util_insert_parm_val( &parm_blkp, GCSET_SERVREQ,
            PARM_SERVICEID, sizeof(char), 0);
```

```
    gc_util_insert_parm_val( &parm_blkp, GCSET_SERVREQ,
            PARM_REQTYPE, sizeof(char), 0);

    gc_util_insert_parm_val( &parm_blkp, GCSET_SERVREQ,
        PARM_ACK, sizeof(char), ISDN_OK);

    gc_util_insert_parm_val( &parm_blkp, GCIS_SET_DLINK,
        GCIS_PARM_DLINK_SAPI, sizeof(char), 0);

    gc_util_insert_parm_val( &parm_blkp, GCIS_SET_DLINK,
        GCIS_PARM_DLINK_CES, sizeof(char), 1);

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SERVREQ,
        GCIS_PARM_SERVREQ_CAUSEVALUE, sizeof(char), NORMAL_CLEARING);

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SERVREQ,
        GCIS_PARM_SERVREQ_USID, sizeof(char), 0x0A);

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SERVREQ,

        GCIS_PARM_SERVREQ_TID, sizeof(char), 0x00);

    gc_util_insert_parm_ref( &parm_blkp, GCIS_SET_SERVREQ,
        GCIS_PARM_SERVREQ_INTERPRETER, sizeof(char), 0x01);

    mode = EV_SYNC;
    ret_val = gc_RespService( GCTGT_GCLIB_CHAN, handle
                parm_blkp, mode);
    if ( ret_val )
    {
        ret_val = gc_ErrorInfo(&t_Info);
        if (ret_val == GC_SUCCESS) {
            printf("gc_ErrorInfo() successfully called\n");
            PrintGC_INFO(&t_Info);
        } else {
            printf("gc_ErrorInfo() call failed\n");

        }
    }
    gc_util_delete_parm_blk( ret_blkp );
    gc_util_delete_parm_blk( parm_blkp );

    return ret_val;
}
```

4.3.3 Supported Service Request Events

Global Call provides the following events for service request support:

- GCEV_SERVICEREQ Event
- GCEV_SERVICERESP Event

4.3.3.1 GCEV_SERVICEREQ Event

The network device receives this event as a Registration Request. The extevtdatap accompanying the event points to a GC_PARM_BLK data structure with the following parameters.

Parameter	Input
GC_PARM_BLK	**set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char **set_id** – GCIS_SET_DCHANCFG **parm_id** – GCIS_PARM_DCHANCFG_AUTOINITFLAG **values** – One of the following: • AUTO_INIT_TERMINAL • NON_INIT_TERMINAL **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_SPID **values** – ASCII digit string limited to the digits 0 to 9 and limited in length to MAX_SPID_SIZE **value_type** – char

4.3.3.2 GCEV_SERVICERESP Event

The GCEV_SERVICERESP event is received by a station device when the network accepts or rejects the registration request. The extevtdatap accompanying the event points to a GC_PARM_BLK data structure with the following parameters.

Parameter	Input
GC_PARM_BLK	**set_id** – GCSET_SERVREQ **parm_id** – PARM_ACK **values** – Any of the Q.931 cause values **value_type** – int **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – 1-8 for BRI when used as a network-side terminal **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – One of the following: • 0 for BRI and PRI • 16 for X.25 packets over D channel **value_type** – char

Parameter	Input
	set_id – GCIS_SET_SERVREQ **parm_id** – GCIS_PARM_SERVREQ_CAUSE **values** – One of the following: • NETWORK_OUT_OF_ORDER • BAD_INFO_ELEM • INVALID_ELEM_CONTENTS • TIMER_EXPIRYPROTOCOL_ERROR **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_USID **values** – range is 01 – FF. 00 signifies default. **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_TID **values** – range is 01 – FF. 00 signifies default. **value_type** – unsigned char **parm_id** – GCIS_PARM_SERVREQ_INTERPRETER (Specifies how the usid and tid values are to be interpreted) **values** – Possible values are: • 0 • 1 **value_type** – unsigned char

4.4 Handling Alarms

Alarm handling using the Dialogic® Global Call API is different depending on the board architecture (Dialogic® DM3 Boards or Dialogic® Springware Boards). The following topics provide information on handling alarms in each architecture:

- Alarm Handling for Dialogic® DM3 Boards
- Alarm Handling for Dialogic® Springware Boards

4.4.1 Alarm Handling for Dialogic® DM3 Boards

When using Dialogic® DM3 Boards, alarms are recognized, and can also be transmitted, on a span (trunk) basis. Once an alarm is detected, all open channels on that span receive a GCEV_BLOCKED event. When the alarm is cleared, open channels receive a GCEV_UNBLOCKED event. Alarm notification only occurs on the first alarm on and the last alarm off. See the *Dialogic® Global Call API Programming Guide* for more information.

The **gc_SetEvtMsk( )** function can be used to mask events on a line device. Using the **gc_SetEvtMsk( )** function on a line device for a time slot sets the mask for the specified time slot only and does not apply to all time slots on the same trunk as is the case when using Dialogic® Springware boards.

The set of Global Call functions that comprise the GCAMS interface for alarm management are supported with the following restrictions:

- Using GCAMS, the application has the ability to set which detected alarms are blocking and non-blocking as described in the *Dialogic® Global Call API Programming Guide*. However, this capability applies on a span basis only. Changing which alarms are blocking and non-blocking for one time slot results in changing which alarms are blocking and non-blocking for all time slots on the span.
- Using the **gc_GetAlarmParm( )** and **gc_SetAlarmParm( )** functions to retrieve and set specific alarm parameters, for example alarm triggers, is not supported.
- The **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )** functions can be used to start and stop the transmission of alarms to the remote side. Table 22 gives the alarms that can be transmitted on ISDN E1 and T1 interfaces.

Table 22. Alarms That Can Be Transmitted on E1 and T1 Interfaces on Dialogic® DM3 Boards

E1 Alarm	T1 Alarm	Description	Equivalent 0x1626 Parameter Value in CONFIG Files Used for Trunk Preconditioning ‡
DEA_REMOTE †	YELLOW †	Remote Alarm Indication (RAI)	2
DEA_UNFRAMED1 †	BLUE †	Alarm Indication Signal (AIS)	1
DEA_SIGNALALL1 †	---	Signaling all 1s Alarm (a multi-frame alarm)	Not applicable
DEA_DISTANTMF †	---	Distant multi-frame alarm	Not applicable

† Defines that can be used in the alarm_number field of the ALARM_FIELD structure when using the **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )** functions to start and stop the transmission of specific alarms.

‡ "Trunk preconditioning" is the ability to place board interface trunks in an alarm state during board initialization. See the Configuration Guide for Dialogic® DM3 Boards for more information.

The following list shows the detected (incoming) alarms that are supported for ISDN on **E1** for Dialogic® DM3 Boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using the **gc_SetAlarmConfiguration( )** function. The default threshold values in some of the alarms that follow can be changed through parameters in the CONFIG file (*.config*). See the Configuration Guide for Dialogic® DM3 Boards for more information.

DTE1_BPVS
: Bipolar violation count saturation. The default threshold value is 255 and the range is 0 to 255.

DTE1_CECS
: CRC4 error count saturation. The default threshold value is 255 and the range is 0 to 255.

DTE1_ECS
: Frame sync bit error count saturation. The default threshold value is 0 and the range is 0 to 255.

DTE1_FSERR
: Received frame sync error

DTE1_FSERROK
Received frame sync error recovered

DTE1_LOOPBACK_CFA
Diagnostic mode on the line trunk

DTE1_LOOPBACK_CFAOK
Diagnostic mode on the line trunk recovered

DTE1_LOS
Received loss of signal

DTE1_LOSOK
Received loss of signal recovered

DTE1_MFSERR
Received multi-frame sync error

DTE1_MFSERROK
Received multi-frame sync error recovered

DTE1_RDMA
Received distant multi-frame alarm

DTE1_RDMAOK
Received distant multi-frame alarm recovered

DTE1_RED†
Received red alarm

DTE1_REDOK
Received red alarm recovered

DTE1_RLOS
Received loss of sync

DTE1_RLOSOK
Received loss of sync recovered

DTE1_RRA†
Received remote alarm

DTE1_RRAOK
Received remote alarm recovered

DTE1_RSA1
Received signaling all 1's

DTE1_RSA1OK
Received signaling all 1's recovered

DTE1_RUA1
Received unframed all 1's

DTE1_RUA1OK
Received unframed all 1's recovered

The following list shows the detected (incoming) alarms that are supported for ISDN on **T1** for Dialogic® DM3 Boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using the **gc_SetAlarmConfiguration( )** function. The default threshold values in some of the alarms following can be changed through parameters in the CONFIG file (*.config*). See the Configuration Guide for Dialogic® DM3 Boards for more information.

DTT1_BPVS
: Bipolar violation count saturation. The default threshold value is 255 and the range is 0 to 255.

DTT1_ECS
: Frame sync bit error count saturation. The default threshold value is 0 and the range is 0 to 255.

DTT1_FERR
: Two out of four consecutive frame bits (F bit) in error. The default threshold value is 0 and the range is 0 to 255.

DTT1_LOOPBACK_CFA
: Diagnostic mode on the line trunk

DTT1_LOOPBACK_CFAOK
: Diagnostic mode on the line trunk recovered

DTT1_LOS
: Initial loss of signal detected

DTT1_LOSOK
: Signal restored

DTT1_OOF
: Out of frame error count saturation. The default threshold value is 0 and the range is 0 to 255.

DTT1_RBL
: Received blue alarm

DTT1_RBLOK
: Received blue alarm restored

DTT1_RCL
: Received carrier loss

DTT1_RCLOK
: Received carrier loss restored

DTT1_RED†
: Received a red alarm condition

DTT1_REDOK
: Red alarm condition recovered

DTT1_RLOS
: Received loss of sync

DTT1_RLOSOK
: Received loss of sync restored

DTT1_RYEL†
: Received yellow alarm

DTT1_RYELOK
Received yellow alarm restored

4.4.2 Alarm Handling for Dialogic® Springware Boards

As described in the *Dialogic® Global Call API Library Reference*, the GCEV_BLOCKED and GCEV_UNBLOCKED events indicate that an alarm condition has occurred or has been cleared, respectively. These events are generated on every opened line device associated with the trunk on which the alarm occurs, if the event is enabled. These events are enabled by default. The application may disable and enable the events by using the **gc_SetEvtMsk( )** function.

If enabling or disabling these events from the board using ISDN, setting the event mask on any line device that represents a time slot will result in setting the mask to the same value on all time slot level line devices on the same trunk. Additionally, setting the event mask on a line device that represents the board will have the same effect (that is, it will set the mask for all time slot level line devices on that trunk).

Alarm notification can be configured for time slot devices using the Global Call Alarm Management System (GCAMS). The set of Global Call functions that comprise the GCAMS interface for alarm management is supported. See the *Dialogic® Global Call API Programming Guide* for more information on GCAMS and the *Dialogic® Global Call API Library Reference* for more information on the GCAMS functions. Alarm notification only occurs on the first alarm on and the last alarm off.

The **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )** functions can be used to start and stop the transmission of alarms to the remote side. Table 23 gives the alarms that can be transmitted on ISDN E1 and T1 interfaces.

Table 23. Alarms That Can Be Transmitted on E1 and T1 Interfaces on Dialogic® Springware Boards

E1 Alarm	T1 Alarm	Description
DEA_REMOTE †	YELLOW †	Remote Alarm Indication (RAI)
DEA_UNFRAMED1 †	BLUE †	Alarm Indication Signal (AIS)
DEA_SIGNALALL1 †	---	Signaling all 1s Alarm (a multi-frame alarm)
DEA_DISTANTMF †	---	Distant multi-frame alarm
† Defines that can be used in the alarm_number field of the ALARM_FIELD structure when using the **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )** functions to start and stop the transmission of specific alarms.		

The following list shows the detected (incoming) alarms that are supported for ISDN on **E1** for Dialogic® Springware Boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using the **gc_SetAlarmConfiguration( )** function.

DTE1_BPVS†
Bipolar violation count saturation

DTE1_BPVSOK
Bipolar violation count saturation recovered

DTE1_CECS†
CRC4 error count saturation

DTE1_CECSOK
CRC4 error count saturation recovered

DTE1_DPM†
Driver performance monitor failure

DTE1_DPMOK
Driver performance monitor failure recovered

DTE1_ECS†
Error count saturation

DTE1_ECSOK
Error count saturation recovered

DTE1_FSERR†
Received frame sync error

DTE1_FSERROK
Received frame sync error recovered

DTE1_LOS†
Received loss of signal

DTE1_LOSOK
Received loss of signal recovered

DTE1_MFSERR†
Received multi-frame sync error

DTE1_MFSERROK
Received multi-frame sync error recovered

DTE1_RDMA†
Received distant multi-frame alarm

DTE1_RDMAOK
Received distant multi-frame alarm recovered

DTE1_RED
Received red alarm

DTE1_REDOK
Received red alarm recovered

DTE1_RLOS†
Received loss of sync

DTE1_RLOSOK
Received loss of sync recovered

DTE1_RRA†
Received remote alarm

DTE1_RRAOK
: Received remote alarm recovered

DTE1_RSA1†
: Received signaling all 1's

DTE1_RSA1OK
: Received signaling all 1's recovered

DTE1_RUA1†
: Received unframed all 1's

DTE1_RUA1OK
: Received unframed all 1's recovered

The following list shows the detected (incoming) alarms that are supported for ISDN on **T1** for Dialogic® Springware Boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using the **gc_SetAlarmConfiguration( )** function.

DTT1_B8ZSD†
: Bipolar eight zero substitution detected

DTT1_B8ZSD
: Bipolar eight zero substitution detected recovered

DTT1_BPVS†
: Bipolar violation count saturation

DTT1_BPVSOK
: BPVS restored

DTT1_DPM†
: Driver performance monitor

DTT1_DPMOK
: Driver performance monitor restored

DTT1_ECS†
: Error count saturation

DTT1_ECSOK
: Error count saturation recovered

DTT1_FERR†
: Frame bit error

DTT1_FERROK
: Frame bit error restored

DTT1_LOS†
: Initial loss of signal detected

DTT1_LOSOK
: Signal restored

DTT1_OOF†
: Out of frame error; count saturation

DTT1_OOFOK
Out of frame restored

DTT1_RBL†
Received blue alarm

DTT1_RBLOK
Received blue alarm recovered

DTT1_RCL†
Received carrier loss

DTT1_RCLOK
Received carrier loss restored

DTT1_RED†
Received a red alarm condition

DTT1_REDOK
Red alarm condition recovered

DTT1_RLOS†
Received loss of sync

DTT1_RLOSOK
Received loss of sync restored

DTT1_RYEL†
Received yellow alarm

DTT1_RYELOK
Received yellow alarm restored

4.5 Handling Errors

In addition to Dialogic® Global Call API cause values that may be retrieved when an error occurs in an ISDN environment, ISDN cause values may also apply. The cause values may originate from several different sources (network, library, or firmware) and are retrieved using either the **gc_ErrorInfo( )** function when <0 is returned or the **gc_ResultInfo( )** function when any event, such as GCEV_TASKFAIL or GCEV_RESTARTFAIL, is returned. For more information, see the "Error Handling" section in the *Dialogic® Global Call API Programming Guide*.

The types of ISDN cause values supported depend on the board architecture used and are described in the following topics:

- ISDN Event Cause Values When Using Dialogic® DM3 Boards
- ISDN Event Cause Values When Using Dialogic® Springware Boards

4.5.1 ISDN Event Cause Values When Using Dialogic® DM3 Boards

ISDN causes comprise two parts: error location and reason. The error location is the upper byte and the reason is the lower byte. For example, the error code NON_ISDN_CAUSE |

CallStateR_Congestion indicates that the error is located in the firmware and the reason for the failure is network congestion. The ISDN error location values when using Dialogic® DM3 Boards are listed in Table 24. See Chapter 11, “ISDN-Specific Event Cause Values” for more information on the individual cause values corresponding to each error location category given in Table 24.

Table 24. ISDN Event Cause Value Sources When Using Dialogic® DM3 Boards

Error Location	Description
Network ERR_ISDN_CAUSE (0x200)	Returned with a GCEV_DISCONNECTED event. The supported values listed in Chapter 11, “ISDN-Specific Event Cause Values” refer to International Telecommunications Union (ITU) Q.931 standards. Not all cause values are universally supported across switch types.
ISDN Library ERR_ISDN_LIB (0x300)	Indicates an ISDN call control library-related cause/error. See Chapter 11, “ISDN-Specific Event Cause Values” for the supported cause values in this category.
Firmware ERR_ISDN_FW (0x100)	Indicates a firmware-related cause/error. Only one cause code of this type is supported when using Dialogic® DM3 Boards, that is, WRONG_MSG_FOR_STATE (0x165).
Firmware NON_ISDN_CAUSE (0xC0)	Indicates a firmware-related cause/error. See Chapter 11, “ISDN-Specific Event Cause Values” for the supported cause values in this category.

4.5.2 ISDN Event Cause Values When Using Dialogic® Springware Boards

ISDN causes comprise two parts: error location and reason. The error location is the upper byte and the reason is the lower byte. For example, the error code ERR_ISDN_FW | ISDN_CHRST_ERR indicates that the error is located in the firmware and the reason for the failure is a channel restart error. The ISDN error location values when using Dialogic® Springware Boards are listed in Table 25. See Chapter 11, “ISDN-Specific Event Cause Values” for more information on the individual cause values supported in each of the categories identified in Table 25.

Table 25. ISDN Event Cause Value Sources When Using Dialogic® Springware Boards

Error Location	Description
Network ERR_ISDN_CAUSE (0x200)	Returned with a GCEV_DISCONNECTED event. Network cause values are listed in the *isdncmd.h* file. The values listed refer to International Telecommunications Union (ITU) Q.931 standards. Not all cause values are universally supported across switch types. See Chapter 11, “ISDN-Specific Event Cause Values” for the supported cause values in this category.
ISDN Library ERR_ISDN_LIB (0x300)	Indicates an ISDN call control library-related cause/error. ISDN library errors are listed in the *isdnerr.h* file. See Chapter 11, “ISDN-Specific Event Cause Values” for the supported cause values in this category.
Firmware ERR_ISDN_FW (0x100)	Indicates a firmware-related cause/error. Firmware errors are listed in the *isdncmd.h* file. See Chapter 11, “ISDN-Specific Event Cause Values” for the supported cause values in this category.

4.6 Controlling the Sending of SETUP_ACK and PROCEEDING

Depending on the board architecture used (Dialogic® DM3 Boards or Dialogic® Springware Boards), the default behavior of the firmware when a SETUP message is received (inbound calls) is different:

- When using Dialogic® DM3 Boards, by default, the firmware automatically sends a SETUP_ACK message if there is no sending complete IE in the received SETUP message. When a SETUP message with a sending complete IE is received, the application must use the **gc_CallAck( )** function to issue the PROCEEDING message to the other side.
- When using Dialogic® Springware Boards, by default, the firmware automatically sends a SETUP_ACK message if there is no sending complete IE in the received SETUP message. When a SETUP message with a sending complete IE is received, the firmware automatically sends the PROCEEDING message to the other side; no intervention by the application is necessary.

A bitmask, that is configurable using the **gc_SetEvtMsk( )** function, and is applicable when using both Dialogic® DM3 and Springware Boards, allows an application developer to modify the default behavior described above. A set of bitmask values can be ORed to mask or unmask the corresponding events. The following bitmask value can be used to mask both the SETUP_ACK and PROCEEDING events:

```
GCMSK_PROC_SEND (0x80)
```

To get full control over the sending of SETUP_ACK and PROCEEDING messages, during startup, an application can issue the following function call:

```
gc_SetEvtMask(..., GCACT_ADDMSK, ..., (GCMSK_PROC_SEND), ...)
```

Then, the application must use **gc_CallAck( )** to send the SETUP_ACK message and **gc_CallAck( )** again to send the PROCEEDING message. Using this technique will ensure that an application is compatible on both Dialogic® DM3 and Springware Boards.

Note: When using Dialogic® Springware Boards, on outbound calls, the GCMSK_SETUP_ACK bitmask value can be used to enable or disable the sending of the GCEV_SETUP_ACK to the application. When using Dialogic® DM3 Boards, GCMSK_SETUP_ACK is **not** supported.

4.7 Handling Glare Conditions

Two common glare conditions and the recommended methods for handling them are described below:

- Receiving a GCEV_TASKFAIL event when using **gc_MakeCall( )** or **gc_SndMoreInfo( )**:
  - When using Dialogic® Springware Boards, while making an outbound call, if the application receives a GCEV_TASKFAIL event (related to some failure) before it receives a response to the SETUP message, the **gc_MakeCall( )** should be considered as having failed. In the case of overlapped sending, the first response is a GCEV_REQMOREINFO

event; any GCEV_TASKFAIL event received subsequently should not be considered a **gc_MakeCall( )** failure.

– When using Dialogic® DM3 Boards, this does not apply since a GCEV_TASKFAIL event is not received when using **gc_MakeCall( )** or **gc_SndMoreInfo( )**. Typically, a GCEV_DISCONNECTED event is received instead.

Note: For both Dialogic® Springware and DM3 Boards, while sending the overlapped digits using **gc_SndMoreInfo( )**, if the answering side accepts or answers the call, depending on the glare, the GCEV_SNDMOREINFO event may not be generated. The application should not wait for this event after getting GCEV_ALERTING, GCEV_PROCEEDING, or GCEV_CONNECTED.

- Receiving a GCEV_DISCONNECTED event when using **gc_AccetpCall( )** or **gc_AnswerCall( )**:

  While accepting or answering an incoming call, if the DISCONNECTED message arrives before the **gc_AcceptCall( )** or **gc_AnswerCall( )** completes, the application does not receive a GCEV_ALERTING or GCEV_ANSWERED event. Instead:

  – When using Dialogic® Springware Boards, the application receives a GCEV_TASKFAIL event with a reason of 0x10F, that is, "Cannot accept event in current state". This is not a serious failure and the application can continue to drop and release the disconnected call and reuse the channel without having to restart it.

  – When using Dialogic® DM3 Boards, the application receives a GCEV_DISCONNECTED event.

4.8 Sending and Receiving Any IE and Any Message

When using Dialogic® Springware and DM3 Boards, the Send Any IE (Information Element) and Send Any Message features provided by the **gc_SetInfoElem( )** and **gc_SndMsg( )** functions are supported by all call control functions, except **gc_ReleaseCall( )**. The Receive Any IE and Receive Any Message features are also supported.

4.9 Using Optional ISDN IEs with Dialogic® DM3 Boards

When writing a telephony application, it is often necessary to work directly with ISDN Information Elements (IEs). Most mandatory IEs are handled automatically with default settings by the Dialogic® Global Call API. By default, Global Call automatically populates the ISDN messages with appropriate mandatory IEs according to parameters given in the various Global Call API calls, based on the various industry standards for basic call control.

However, some optional and proprietary IEs contained in inbound and outbound messages are not available in parsed format for application access through the standard API call control functions. In those cases, the application must take a direct role in formatting, parsing, and transmitting the appropriate IEs.

The following sections describe:

- Creating New IEs
- Modifying Existing IEs

- Creating New Messages
- Receiving IEs

4.9.1 Creating New IEs

A new IE is added to the outbound message buffer for an outbound call using **gc_SetInfoElem( )** before the call is initiated (and the SETUP message sent) with **gc_MakeCall( )**.

First, a definition of the IE to be sent should be obtained. One method for obtaining an IE definition is to run an ISDN trace on a known-working system as a call SETUP message is transmitted. Alternatively, the appropriate ISDN specification can be consulted for the IE values.

The IE data values are built into an IE_BLK data structure, and then passed as a parameter to **gc_SetInfoElem( )**. The IE_BLK structure contains the data and length of data:

```
typedef struct {
   short   length;              /* must be less than MAXLEN_IEDATA */
   char    data[MAXLEN_IEDATA]; /* application defined data */
} IE_BLK, *IE_BLK_PTR;
```

Copy (**memcpy( )**) into the data field of the structure the number of bytes contained in the IE. A second structure is also needed for **gc_SetInfoElem( )**:

```
typedef struct {
   GCLIB_IE_BLK *gclib;
   void         *cclib;
} GC_IE_BLK, *GC_IE_BLKP;
```

In this case, gclib is set to NULL, and cclib is set to point to the IE_BLK with the information element already loaded in it. At this point, **gc_SetInfoElem(LINEDEV, GC_IE_BLKP)** can be called. **gc_SetInfoElem( )** requires the Global Call line device for the opened channel and a pointer to the structure containing the IE.

Finally, **gc_MakeCall( )** can be called. The SETUP message it generates will contain the IE. Note that any outbound message under application control can have optional IEs attached using this method.

4.9.2 Modifying Existing IEs

It is possible to change an existing IE in the SETUP message. To do this, follow the same steps outlined above to build a replacement IE and call **gc_SetInfoElem( )** with it. The replacement IE will appear, overwriting (but not duplicating) the original.

This capability is especially useful for setting the presentation indicator in the calling number IE or the called/calling number types in their respective IEs. (See Figure 49, “IE Flow”, on page 168.) Note that you must replace the entire IE, not just the subfields. Also, note that you need to follow two ISDN rules:

- All information elements you add to the optional information element buffer need to be in numerical order by IE type.
- ISDN allows only one IE of any type per message.

4.9.3 Creating New Messages

A complete message can also be constructed and sent at the appropriate time in a call. This would typically be a USER INFORMATION message, which is used to send application defined data pertaining to an ongoing call across the D channel. Review the header files to see which ISDN messages are defined and allowed for the **gc_SndMsg( )** function call.

For example, you can put together a USER INFORMATION message with a single user-to-user IE. The IE is populated with 10 bytes of data (in this case, the ASCII digits 0-). The IE_BLK is built as described above, with the user-to-user IE identifier 0x7e, a value for the protocol discriminator field, the data count for the IE, and the user-defined data itself.

Once the IE is built and wrapped in a GC_IE_BLK, it is sent to the network using **gc_SndMsg(LINEDEV, CRN, SndMsg_UsrInformation, GC_IE_BLKP)**. Parameters for the API call are the Global Call line device for the opened channel, the call reference number assigned to the call, an indicator that the IE should be sent within a USER INFORMATION message, and a pointer to the structure containing the IE.

4.9.4 Receiving IEs

Looking at any IE that has been received as part of an incoming message is primarily a matter of calling **gc_GetSigInfo( )** at the appropriate time.

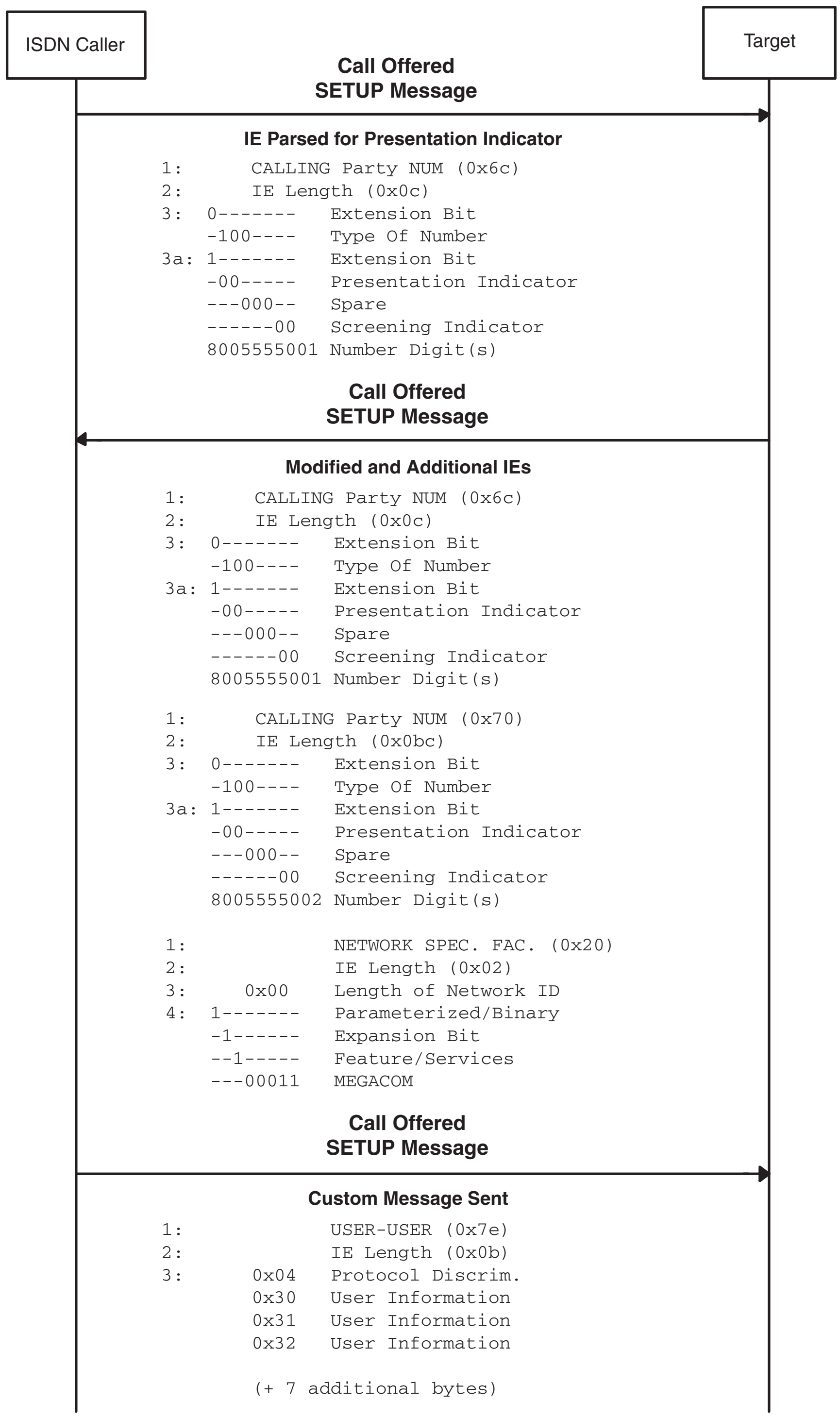
For example, the following steps show how the presentation restriction field in a calling number IE is read for a GCEV_OFFERED (call offered) event, resulting from the arrival of a SETUP message:

- Set RECEIVE_INFO_BUF using **gc_SetParm( )** after **gc_OpenEx( )** on each channel/linedev.
- On getting a call offered event, null out an IE_BLK structure.
- Use the IE_BLK structure for a call to **gc_GetSigInfo( )** to fetch all IEs contained in the SETUP message. Global Call-related parameters linedevice and metaevent are also needed, but are readily available.
- Process the returned data. General format of data is:
  - IE type
  - Count of remaining data bytes
  - The data itself
  - The next IE type, and so on

If the data format is known, it is possible to parse out each individual IE and its data using index/pointer arithmetic. Select the appropriate IE structure format by IE type and extract the required fields and associated values by using comparisons and/or bitmasks according to the definition of each individual IE.

For other IEs, ISDN specifications should be consulted. Be aware that some of the IEs use the most significant bit of one byte of the IE field to indicate the existence of an additional byte.

Figure 49. IE Flow



4.10 Using Overlap Send

Note: The protocols that support overlapped sending are Net5 and QSIG. The 4ESS, 5ESS, NI2, DMS, and NTT protocols do not support overlapped sending.

When using Dialogic® Springware Boards, to activate the overlap send feature that prevents the automatic sending of a Sending Complete IE within the SETUP message, parameter 0024 in the firmware PRM file must be set to a value that includes the bit represented by the value 08H. See Table 26, "Modifiable Protocol Parameters for Dialogic® Springware Boards", on page 192 for more information.

When using Dialogic® DM3 Boards, to activate the overlap send feature that prevents the automatic sending of a Sending Complete IE within the SETUP message, the following modifications should be made to the CONFIG file for the desired outbound protocol variant requiring overlap send support. The **CalledNumberCount** parameter, which has a default value of zero, should be set to a large positive value. For example, in the ISDN Protocol Variant Definitions section of the CONFIG file being used, change:

```
Variant CalledNumberCount 99
```

Note: You can have more than one **CalledNumberCount** setting per board; in order to do so, create a new Variant Define and apply that define using the `defineBSet` command in the respective TSC section.

See the configuration information for Dialogic® DM3 products provided with the Dialogic® Software Release for more information on how to perform the changes outlined above.

A **gc_MakeCall( )** function call that specifies fewer digits than the **CalledNumberCount** results in the sending of a SETUP message that does **not** contain a Sending Complete IE. If more digits are specified, the Sending Complete IE is included in the SETUP message.

Note: Changes to the CONFIG file for a particular protocol are automatically updated in the corresponding FCD file when the firmware is downloaded to the board. For more information, see the configuration guide for Dialogic® DM3 products.

4.11 Using Direct Layer 2 Access

When using Dialogic® Springware Boards, to activate layer 2 access, parameter 0024 in the firmware PRM file must be set to a value that includes the bit represented by the value 01H. See Table 26, "Modifiable Protocol Parameters for Dialogic® Springware Boards", on page 192 for more information.

When using Dialogic® DM3 Boards, direct layer 2 access is supported on a per trunk basis. Direct layer 2 access is enabled by including the following command in the appropriate [CSS.x] section of the CONFIG file, where x identifies a specific trunk (span):

```
Setparm=0x9,1
```

If this command is not included, direct layer 2 access is disabled. Also, using a 0 instead of a 1 in the command above disables direct layer 2 access.

Note: Changes to the CONFIG file for a particular protocol are automatically updated in the corresponding FCD file when the firmware is downloaded to the board. For more information, see the configuration guide for Dialogic® DM3 products.

Global Call supports direct layer 2 access using the **gc_GetFrame( )** and **gc_SndFrame( )** functions.

4.12 Getting D Channel Status

When using Dialogic® DM3 Boards, a GCEV_D_CHAN_STATUS event is always generated once the board device is initialized and the initial D channel status is known. The resulting value associated with the event indicates this initial D channel status. Any subsequent change in the D channel status is also notified by means of GCEV_D_CHAN_STATUS event. When using Dialogic® Springware Boards, when the initial D channel status was UP, no initial event was generated. When using Dialogic® DM3 Boards, an initial event is always generated, regardless of the initial status of the D channel.

On download, by default both the trunk and channels are out of service. When the first **gc_OpenEx( )** is executed on a device, the trunk (D channel) and the channel (B channel) associated with the device are placed into service (trunk in service, channel idle). Although the channel is IDLE, calls cannot be received or processed until **gc_WaitCall( )** is issued. When the application uses **gc_Close( )** to close the channel, the channel returns to out of service, but the trunk remains in service.

The application should use the **gc_ResultValue( )** function to find the reason (UP or DOWN) associated with the GCEV_D_CHAN_STATUS event. A reason of UP indicates that the D channel is active, and the **gc_GetFrame( )** and **gc_SndFrame( )** functions can be used to get or send frames respectively. The **gc_GetLinedevState( )** function can be used to retrieve the status of the line device. See the *Dialogic® Global Call API Library Reference* for more information.

4.13 Controlling B Channel Status

When using Dialogic® DM3 Boards, the initial B channel state (in service or out of service) is controlled by a CHP parameter (parameter 0x1311) in the CONFIG file. By default, all channels are out of service after the board firmware is downloaded. Channels are brought into service when devices are opened using the **gc_Open( )** function.

4.14 B Channel Negotiation

When using Dialogic® DM3 Boards, the Dialogic® Global Call API supports B channel negotiation for ISDN PRI protocols. To understand the level of support, it is important to understand the related channel states in the firmware:

Busy
: A call is already using the channel.

Barred
: No user application has issued a **gc_WaitCall( )** on the channel.

Out-Of-Service
: The channel is out of service.

The following terms are used as a convenience to describing B channel negotiation below:

Unavailable
: A channel is in a Busy, Barred, or Out-Of-Service state.

Available
: A channel is not in a Busy, Barred, or Out-Of-Service state.

The B channel negotiation behavior is summarized as follows:

- If a new incoming call is received with its CHANNEL_ID_IE set to PREFERRED (not EXCLUSIVE) identifying a specific channel and the specified channel is currently *unavailable*, the firmware tries to find an *available* channel for the call before the call is presented to the host application. If the firmware cannot find an *available* channel, it rejects the incoming call with a cause value set to 54 (incoming call barred).
- If a new incoming call is received with its CHANNEL_ID_IE set to ANY_CHANNEL, the firmware tries to find an *available* channel for the call before the call is presented to the host application. If the firmware cannot find an *available* channel, it rejects the incoming call with a cause value set to 54 (incoming call barred).
- If a new incoming call is received with its CHANNEL_ID_IE specifying an EXCLUSIVE channel and the specified channel is currently Barred, the firmware rejects the incoming call with a cause value set to 54 (incoming call barred). If the specified channel is currently Out-Of-Service or Busy, the firmware rejects the incoming call with a cause value of 34 (no circuit available) or a cause value of 44 (requested channel not available) depending on the protocol switch type and switch side in use.

4.15 Call Progress Analysis When Using Dialogic® DM3 Boards

Pre-connect *call progress* and post-connect *call analysis*, collectively know as Call Progress Analysis (CPA), are supported as described in the *Call Control* chapter of the *Dialogic® Global Call API Programming Guide*.

Note: Call Progress Analysis (CPA) requires that a voice device be attached to the network device. This can be achieved by specifying the voice device when issuing the **gc_OpenEx( )** function (a feature not supported in earlier releases) or opening a network device only and subsequently attaching a voice device using the **gc_AttachResource( )** function. See the **gc_OpenEx( )** and **gc_AttachResource( )** function reference pages in the *Dialogic® Global Call API Library Reference*.

Default values for CPA parameters are defined in the CONFIG file for the board. The parameters include:

- **CallProgress**
- **CaHdgLoHiGl**
- **CaAnsdglPSV**
- **CaRingingSet**
- **CaBusySet**
- **CaSitSet**
- **CaFaxSet**
- **CaPvdId**
- **CaPamdId**
- **CaSignalTimeout**
- **CaAnswerTimeout**
- **CaPvdTimeout**

See the Configuration Guide for your product for more information about modifying the values of these parameters.

Note: The default value for the **CallProgress** parameter is Y. This value allows the different CPA options to be enabled or disabled, but does not actually specify that CPA will be used. It is the application's responsibility to specify when CPA is used.

The following options are available to the user:

- If an application does not explicitly specify CPA, then CPA is OFF by default. This is irrespective of the **CallProgress** = Y parameter in the CONFIG file.
- If an application uses Global Call functions (for example, **gc_MakeCall( )**, **gc_SetConfigData( )**, or **gc_SetParm( )**) to specify CPA, the default values of CPA parameters in the board's CONFIG file are used.
- To use CPA parameter values that are different than the values specified in the CONFIG file, a user can:
  - Use the Global Call API to explicitly specify the desired values.

– Change the values in the CONFIG file and redownload the firmware to the board.

When using Dialogic® DM3 Boards that support flexible routing configurations, the **dx_dial( )** method for call analysis continues to be supported. Both pre-connect call progress and post-connect call analysis are available. See the *Dialogic® Voice API Programming Guide* for more information on call analysis and the *Dialogic® Voice API Library Reference* for details on the relevant API functions.

4.16 Implementing Call Hold and Retrieve

The Dialogic® Global Call API supports call hold and retrieve functionality using the following API functions:

- **gc_HoldAck( )**
- **gc_HoldCall( )**
- **gc_HoldRej( )**
- **gc_RetrieveAck( )**
- **gc_RetrieveCall( )**
- **gc_RetrieveRej( )**

See the *Dialogic® Global Call API Library Reference* for more detail on the Global Call functions mentioned above.

Call hold and retrieve is supported in the Proceeding, Accepted, Alerting, and Connected states. The ability to do a second **gc_MakeCall( )** in any of these states is supported. One limitation is that a call can only be retrieved on the same channel from which it was put on hold.

The hold and retrieve functionality described in this section supports the sending and receiving of any IE with the Hold and Retrieve requests using the **gc_SetInfoElem( )** and **gc_GetSigInfo( )** functions.

The level of support for Dialogic® Springware and DM3 Boards is as follows:

- When using Dialogic® DM3 Boards, call hold and retrieve functionality is supported for the following ISDN protocols:
  - 4ESS
  - 5ESS
  - DMS
  - NET5
  - NI2
  - NTT
  - QSIG
- When using Dialogic® Springware Boards, call hold and retrieve functionality is supported for the following protocols:
  - BRI
  - PRI NTT

– PRI DPNSS
– PRI QSIG

See the *Call State Models* chapter of the *Dialogic® Global Call API Programming Guide* for more information. See also Section 3.1.26, "Call Hold and Retrieve Scenarios", on page 94 for more information on how call hold and transfer is implemented when using ISDN protocols on Dialogic® DM3 Boards.

4.17 Using Dynamic Trunk Configuration

When using Dialogic® DM3 Boards, the Dialogic® Global Call API provides the ability to perform the following dynamic configuration operations at run time:

- Setting the ISDN Protocol Mode for a Trunk
- Setting the Line Type and Coding for a Trunk
- Specifying the Protocol for a Trunk

Note: The **gc_SetConfigData( )** function can be used on a board device to perform these operations. However, it is the application's responsibility to handle all active calls on the trunk, and terminate them if necessary. In addition, the **gc_ResetLineDev( )** function may be issued on all channels (time slots) prior to issuing **gc_SetConfigData( )** to prevent incoming calls. If there are any active calls present at the time the **gc_ResetLineDev( )** or **gc_SetConfigData( )** function is issued, they are gracefully terminated internally. The application does not receive GCEV_DISCONNECTED events when calls are terminated in this manner.

4.17.1 Setting the ISDN Protocol Mode for a Trunk

Note: This feature is only applicable when using Dialogic® DM3 Boards.

The **gc_SetConfigData( )** function can be used to change the ISDN protocol mode (user or network) without having to re-download the board firmware. This means that it is not necessary to stop processing calls while the settings are changed on a single trunk.

The **gc_SetConfigData( )** function uses a GC_PARM_BLK structure that contains the configuration information. The GC_PARM_BLK is populated using the **gc_util_insert_parm_val( )** function.

To configure User or Network mode, use the **gc_util_insert_parm_val( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_USER_NETWORK
- **data_size** = 4 (integer)
- **data** = either 0 (User mode) or 1 (Network mode)

Once the GC_PARM_BLK has been populated with the desired values, the **gc_SetConfigData( )** function can be issued to perform the configuration. The parameter values for the **gc_SetConfigData( )** function are as follows:

- **target_type** = GCTGT_CCLIB_CHAN
- **target_id** = the trunk line device handle, as obtained from **gc_OpenEx( )** with a **devicename** string of ":N_dtiBx:P_ISDN", which can also optionally include a voice device
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the **gc_util_insert_parm_val( )** utility function
- **time_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = GCUPDATE_IMMEDIATE
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution

Notes: ***1.*** The application must include the *dm3cc_parm.h* header file when using this feature.

2. The configuration changes made by issuing **gc_SetConfigData( )** are not persistent, that is, the CONFIG and FCD files are not updated.

Example

In the following example, assume that **ldev** is a LINEDEV-type variable, properly initialized by a successful call to **gc_OpenEx( )**.

```
GC_PARM_BLKP ParmBlkp = NULL;
long id;
if (sr_waitevt(-1) >= 0) {
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype()) {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s \n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device %s, Error=%s\n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()),
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevttype(),
                  ATDV_NAMEP(sr_getevtdev()));
            break;
    }

}
```

4.17.2 Setting the Line Type and Coding for a Trunk

Note: This feature is only applicable when using Dialogic® DM3 Boards.

The **gc_SetConfigData( )** function can be used on a board device to reconfigure the line type for a trunk. The **gc_SetConfigData( )** function uses a GC_PARM_BLK structure that contains the configuration information. The GC_PARM_BLK is populated using the **gc_util_insert_parm_val( )** function.

To configure the *line type*, use the **gc_util_insert_parm_val( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_LINE_TYPE
- **data_size** = sizeof(int)
- **data** = One of the following values:
  - Enum_LineType_dsx1_D4 - D4 framing type, Superframe (SF)
  - Enum_LineType_dsx1_ESF - Extended Superframe (ESF)
  - Enum_LineType_dsx1_E1 - E1 standard framing
  - Enum_LineType_dsx1_E1_CRC - E1 standard framing and CRC-4

To configure *coding type*, use the **gc_util_insert_parm_val( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_CODING_TYPE
- **data_size** = sizeof(int)
- **data** = One of the following values:
  - Enum_CodingType_AMI - Alternate Mark Inversion
  - Enum_CodingType_B8ZS - Modified AMI used on T1 lines
  - Enum_CodingType_HDB3 - High Density Bipolar of Order 3 used on E1 lines

Once the GC_PARM_BLK has been populated with the desired values, the **gc_SetConfigData( )** function can be issued to perform the configuration. The parameter values for the **gc_SetConfigData( )** function are as follows:

- **target_type** = GCTGT_CCLIB_NETIF
- **target_id** = the trunk line device handle, as obtained from **gc_OpenEx( )** with a **devicename** string of ":N_dtiBx:P..."
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )**

- **time_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = GCUPDATE_IMMEDIATE
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution

The application receives one of the following events:

- GCEV_SETCONFIGDATA to indicate that the request to dynamically change the line type and/or coding has been successfully initiated.
- GCEV_SETCONFIGDATAFAIL to indicate that the request to dynamically change the line type and/or coding failed. More information is available from the GC_RTCM_EVTDATA structure associated with the event.

The following code example shows how to dynamically configure a T1 trunk to operate with the Extended Superframe (ESF) line type and the B8ZS coding type.

```
GC_PARM_BLKP ParmBlkp = NULL;
long id;

/* configure Line Type = Extended Superframe for a T1 trunk */
gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_LINE_TYPE, sizeof(int),
                        Enum_LineType_dsx1_ESF);

/* configure Coding Type = B8ZS for a T1 trunk */
gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_CODING_TYPE, sizeof(int),
                        Enum_CodingType_B8ZS);

gc_SetConfigData(GCTGT_CCLIB_NETIF, bdev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
                 gc_util_delete_parm_blk(ParmBlkp);

if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s
                   \n",((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                   ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device
                   %s, Error=%s\n",((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                   ATDV_NAMEP(sr_getevtdev()),
            ((GC_RTCM_EVTDATA *)(meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevttype(),
                   ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
```

4.17.3 Specifying the Protocol for a Trunk

Note: This feature is only applicable when using Dialogic® DM3 Boards.

The protocol used by a trunk can be dynamically configured after devices have been opened using the **gc_SetConfigData( )** function. All channels on the affected trunk inherit the newly selected protocol.

The **gc_SetConfigData( )** function uses a GC_PARM_BLK structure that contains the configuration information. The GC_PARM_BLK is populated using the **gc_util_insert_parm_ref( )** function.

To configure the *protocol*, use the **gc_util_insert_parm_ref( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = GCSET_PROTOCOL
- **parmID** = GCPARM_PROTOCOL_NAME
- **data_size** = strlen("<protocol_name>"), for example strlen("4ESS")
- **data** = "<protocol_name>", for example, "4ESS" (a null-terminated string). For ISDN protocols, the protocol name must be one of the supported protocols listed in the CONFIG file that corresponds to the PCD/FCD file that is downloaded. Only protocols of the same line type can be selected, that is, if the trunk is of line type E1, then only a protocol variant that is valid for E1 can be selected.

Once the GC_PARM_BLK has been populated with the desired values, the **gc_SetConfigData( )** function can be issued to perform the configuration. The parameter values for the **gc_SetConfigData( )** function are as follows:

- **target_type** = GCTGT_CCLIB_NETIF
- **target_id** = the trunk line device handle, as obtained from **gc_OpenEx( )** with a **devicename** string of ":N_dtiBx:P..."
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_ref( )**
- **time_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = GCUPDATE_IMMEDIATE
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution

The application receives one of the following events:

- GCEV_SETCONFIGDATA to indicate that the request to dynamically change the protocol has been successfully initiated.

- GCEV_SETCONFIGDATAFAIL to indicate that the request to change the protocol has failed. More information is available from the GC_RTCM_EVTDATA structure associated with the event.

The following code example shows how to dynamically configure a T1 trunk to operate with the 4ESS protocol.

```
static int MAX_PROTOCOL_LEN=20;
GC_PARM_BLKP ParmBlkp = NULL;
long id;
char protocol_name[]="4ESS";

gc_util_insert_parm_ref(&ParmBlkp, GCSET_PROTOCOL, GCPARM_PROTOCOL_NAME,
strlen(protocol_name)+1, protocol_name);

gc_SetConfigData(GCTGT_CCLIB_NETIF, bdev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
gc_util_delete_parm_blk(ParmBlkp);

if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);

    switch(sr_getevttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s
                    \n",((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                    ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device
                    %s, Error=%s\n",((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                    ATDV_NAMEP(sr_getevtdev()),
                    ((GC_RTCM_EVTDATA *)(meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevttype(),
                    ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
```

4.18 Retrieving Continuity Check IE

The Dialogic® Global Call API supports the retrieval of the Continuity Check IE (0x53) that is part of Codeset 6 for the 4ESS protocol.

The receipt of a FACILITY message containing this IE triggers an unsolicited GCEV_FACILITY event. Upon receipt of this event, the application can call the **gc_GetSigInfo( )** function to retrieve the IE information (continuity indication) included with the FACILITY message.

4.19 Sending and Receiving DPNSS End to End Messages

When using Dialogic® DM3 Boards, the Dialogic® Global Call API provides the ability to send and receive the entire raw Digital Private Network Signaling System (DPNSS) end to end message (EEM) using API control. A generic mechanism enables the user to add DPNSS supplementary services (like single/dual channel transfer services, call diversion, and call waiting) without needing outside support for those services first. This feature is only supported on ISDN DPNSS loads.

4.19.1 Overview

This feature enables the application to:

- Enable GCEV_EXTENSION through **gc_SetConfigData( )** (for enabling the event).
- Send raw DPNSS EEM through **gc_SndMsg( )** with the specified message type (for sending the event).
- Receive raw DPNSS EEM through GCEV_EXTENSION event on Dialogic® DM3 Boards (for receiving the event).

The user has the ability to send and receive raw EEM frames. The user can extract the content of the EEM message and take the appropriate action when he/she receives any of the messages. The API is allowed in any intermediate call state. A majority of DPNSS supplementary services can be supported, and the user does not need to request outside support for every new service that is being planned for the future.

EEM frames are of two types:

- EEM(I) - an end to end message (incomplete)
- EEM(C) - an end to end message (complete)

An EEM(C) is typically used, but if the size of the message exceeds 45 bytes in length, it can be split up into multiple EEM(I) messages, with a final piece of the message completed by an EEM(C).

Note: The application tracks the receipt of the various EEM(I) frames and reassembles them together to form the entire final EEM(C) message.

Parameters

The feature is implemented using the Global Call **gc_SetConfigData( )** function and the GCTGT_CCLIB_CHAN parameter set. Extension event IDs define the receive raw DPNSS EEM through the GCEV_EXTENSION event. This unsolicited event can be enabled or disabled through **gc_SetConfigData( )**. The ISDN *dm3cc_param.h* header file includes the following:

- Extension ID: DM3CC_EXT_EVT_RAWEEM
- Bit mask: EXTENSIONEVT_RAWEEM - Use to enable or disable the GCEV_EXTENSION event for DPNSS Raw EEM

- Set ID: CCSET_RAWEEM
- PARMID: CCPARM_RAWEEM_DATA

The ISDN *isdndef.h* header file includes SndMsg_RawEEM for the application to send raw EEM through **gc_SndMsg( )**.

Generated Events

The **gc_SetConfigData( )** function is issued to enable this functionality, and the following notification event is generated for the application:

EXTENSIONEVT_RAWEEM
: Use to enable or disable the GCEV_EXTENSION event for DPNSS raw EEM

The **gc_SndMsg( )** function is issued and the following notification events may be generated for the application:

CCSET_RAWEEM
: Receives raw EEM

GCEV_TASKFAIL
: Indicates failure, for example, in case the information element (IE) has state change information in that the raw data contains an invalid IE or the raw data is 45 bytes

Error Codes

The following success code is generated by the **gc_SetConfigData( )** function:

GC_SUCCESS
: Success. The signal type change has been implemented.

The following error code is generated by the **gc_SndMsg( )** function:

GCEV_TASKFAIL
: Task failed. The firmware will return Std_MsgError if the call state is not transferring because there is an invalid IE (call state changing), or the raw data is 45 bytes.

4.19.2 Enabling/Disabling GCEV_EXTENSION Event

For the **gc_SetConfigData( )** function, the bit mask (EXTENSIONEVT_RAWEEM) is saved for use later during GCEV_EXTENSION event generation. The **gc_SetConfigData( )** is set on a channel basis and has the target type set as GCTGT_CCLIB_CHAN. For example:
gc_SetConfigData(GCTGT_CCLIB_CHAN.EXTENSIONEVT_RAWEEM)

4.19.3 Sending and Receiving Raw DPNSS EEM

The **gc_SndMsg(MsgType, GC_IE_BLK)** function is used to send the raw DPNSS EEM like other DPNSS Supplementary Services (for example, Intrusion(SndMsg_Intrude), Diversion(SndMsg_Divert), NSI (SndMsg_NSI), etc.). The general procedures are as follows:

- To send an End to End Complete message, call the **gc_SndMsg( )** function with the msg_type parameter set to SndMsg_RawEEM. The first byte of the data portion (i.e., ie_Blk.data[0]) must contain 0x22 to indicate that it is an EEM(C) message. To receive the message, enable the GCEV_EXTENSION event.
- To send an End to End Incomplete message, call the **gc_SndMsg( )** function, with the msg_type parameter set to SndMsg_RawEEM. The first byte of the data portion (i.e., ie_Blk.data[0]) must contain 0x23 to indicate that it is an EEM(I) message. To receive the message, enable the GCEV_EXTENSION event.

The message is successfully received if no GCEV_TASKFAIL event is received at the user application.

Notes: ***1.*** The first byte in the GC_IE_BLK is the spec-defined Message ID for an EEM(I) or EEM(C) message.

2. Certain supplementary information strings that may affect the firmware call state are not allowed in the raw EEM payload. Specifically not allowed are the HOLD-REQ string or 60B, and the RECON string or 61. If either of these strings is present, the application will receive a GCEV_TASKFAIL event.

3. The total length of the raw EEM payload allowed is 45 bytes: 1 byte specifies the EEM type, which is EEM(C) or EEM(I), and 44 bytes are allowed for supplementary information strings encoded using the Backus Naur format and conforming to the DPNSS standard BTNR 188.

4.19.4 Sample Code

The following are code samples for sending raw EEM and receiving raw EEM.

To Send Raw EEM

```
int send_message(CRN crn)
{

 int      gc_err;          /* GlobalCall Error Code */
    int      cclibid;         /* Call Control library ID */
    long     cclib_err;       /* Call Control Error Code */
    char     *msg;            /* Error Message */
    LINEDEV  ldev;            /* Line device */
 char      str[MAX_STRING_SIZE];

 GC_IE_BLK gcIEBlk;
 IE_BLK ie_Blk;

 memset((unsigned char *)&ie_Blk, 0, sizeof(IE_BLK));

 gcIEBlk.gclib = NULL;
 gcIEBlk.cclib = &ie_Blk;
 ie_Blk.length =  7;       //length of the raw DPNSS EEM data
  /* EEM(C) = 0x22, EEM(I) = 0x23 */
 ie_Blk.data[0] = 0x22;   // raw DPNSS EEM data
```

```
 ie_Blk.data[1] = '*';   // raw DPNSS EEM data
 ie_Blk.data[2] = '1';   // raw DPNSS EEM data
 ie_Blk.data[3] = '1';   // raw DPNSS EEM data
 ie_Blk.data[4] = '0';   // raw DPNSS EEM data
 ie_Blk.data[5] = 'B';   // raw DPNSS EEM data
 ie_Blk.data[6] = '#';   // raw DPNSS EEM data

      if(gc_CRN2LineDev(crn, &ldev) != GC_SUCCESS) {
        gc_ErrorValue(&gc_err, &cclibid, &cclib_err);
        gc_ResultMsg(cclibid, cclib_err, &msg);
  sprintf(str, "Error on Device handle : 0x%lx ",ldev);
  printandlog(0, GC_APICALL, NULL, str, 0);
        return(cclib_err);
    }

    if(gc_SndMsg(ldev, crn, SndMsg_RawEEM, &gcIEBlk) != GC_SUCCESS) {
        gc_ErrorValue(&gc_err, &cclibid, &cclib_err);
        gc_ResultMsg(cclibid, cclib_err, &msg);
        sprintf(str, "Error on Device handle : 0x%lx ",ldev);
  printandlog(0, GC_APICALL, NULL, str, 0);
        return(cclib_err);
    }
    return 0 ;
}
```

To Enable the GCEV_EXTENSION Event to Receive Raw EEM Events

```
int EnableRawEEMInformation(int DeviceHdl)
{
 GC_PARM_BLKP pParmBlock = NULL;
 long requestID;
 char      str[MAX_STRING_SIZE];

 int iRetCode = gc_util_insert_parm_val(&pParmBlock, CCSET_EXTENSIONEVT_MSK,
  GCACT_ADDMSK, sizeof(long), EXTENSIONEVT_RAWEEM);

 int rc = gc_SetConfigData(GCTGT_CCLIB_CHAN, DeviceHdl, pParmBlock,0,
  GCUPDATE_IMMEDIATE, &requestID, EV_ASYNC);

 if(rc != GC_SUCCESS) {
  sprintf(str, "failed to set evt mask");
  printandlog(0, GC_APICALL, NULL, str, 0);
  return GC_ERROR;
 } else {
  sprintf(str, "gc_SetConfigData() called - Raw EEM event  reception enabled");
  printandlog(0, GC_APICALL, NULL, str, 0);
 }

 gc_util_delete_parm_blk(pParmBlock);

 return 0;
}
```

To Receive Raw EEM and Extract Raw DPNSS Data

```
void process_event(void)
{ ....
  ....
  ....
 switch (evttype)
 {
  case GCEV_EXTENSION:
```

```
   ExtractDPNSSInfo(pline, &metaevent);
   break;

  }
}

void ExtractDPNSSInfo(struct channel *pline,METAEVENT *metaeventp)
{
 GC_PARM_BLKP gcParmBlkp = NULL;
 GC_PARM_DATAP t_gcParmDatap = NULL;
 EXTENSIONEVTBLK *ext_evtblkp = NULL;

 GC_IE_BLK * t_gcIEBlk = NULL;
 IE_BLK * ie_blk = NULL;
 char rawData[100];
 char str[MAX_STRING_SIZE];
 int i=0;

 ext_evtblkp = (EXTENSIONEVTBLK *)metaeventp->extevtdatap;
 gcParmBlkp = &ext_evtblkp->parmblk;

 sprintf(str, "Received GCEV_EXTENSION event with ExtID = 0x%x",ext_evtblkp->ext_id);
 printandlog(0, GC_APICALL, NULL, str, 0);
 while (t_gcParmDatap = gc_util_next_parm(gcParmBlkp, t_gcParmDatap))
 {
 switch (t_gcParmDatap->set_ID)
  {
   case CCSET_RAWEEM:
    switch(t_gcParmDatap->parm_ID)
    {
     case CCPARM_RAWEEM_DATA:
         t_gcIEBlk =  (GC_IE_BLK *)t_gcParmDatap->value_buf;
      ie_blk =  t_gcIEBlk -> cclib;
      memcpy(rawData, ie_blk->data,ie_blk->length);

      sprintf(str, "RAWEEM_DATA : length = %d\n", ie_blk->length);
      printandlog(0, GC_APICALL, NULL, str, 0);
      memset(str, 0, MAX_STRING_SIZE);

      for (i=0; i < ie_blk->length; i++)
      {
       if((i!=0) && (isascii(rawData[i]))) {
        printf(str, "%c ", rawData[i]);

        fprintf(port[0].log_fp, "%c ", rawData[i]);
       }

       else {
        printf(str, "%02X ", rawData[i]);
        fprintf(port[0].log_fp, "%02X ", rawData[i]);
       }
      }
      printf("\n");
      fprintf(port[0].log_fp, "\n ");

     break;
     default:
      sprintf(str, "Unknown PARM ID");
      printandlog(0, GC_APICALL, NULL, str, 0);
     break;
    }
    break;
   default:
    sprintf(str, "Unknown SET ID");
    printandlog(0, GC_APICALL, NULL, str, 0);
```

```
      break;
     }
   }

}
```

ISDN Protocols 5

This chapter describes the Integrated Services Digital Network (ISDN) protocols supported by the Dialogic® Global Call API, the firmware and parameter files for each protocol, and protocol parameters. Topics include:

5.1 Basic Rate Interface

The Basic Rate Interface (BRI) and Dialogic® Global Call API support for BRI is described in the following topics:

- Hardware Support for BRI
- Features of BRI
- Typical BRI Applications

5.1.1 Hardware Support for BRI

There are two types of Dialogic® BRI Boards, BRI/SC and BRI/2:

- The Dialogic® BRI/SC Boards allow individual routing of up to 32 B channels (voice/data channels) and 16 D channels (signaling channels) to any of the application-selectable SCbus time slots using the SCbus distributed switching capability. B channel traffic may be routed from the ISDN network or local station set device to and from the SCbus. Dialogic® BRI/SC Boards can be used in either a Windows® or a Linux operating environment.

  The Global Call BRI/SC protocol implementations comply with the North American standard ISDN BRI, Euro-ISDN protocol for BRI, and the INS64 standard used in Japan.
- The Dialogic® BRI/2 Boards emulate two standard BRI station sets with display, and are designed to support the Euro-ISDN protocol. The Dialogic® BRI/2 Boards provide analog voice processing, via the Dialogic® Voice API (see Note 1 below) and the ISDN API, and support many enhanced ISDN features. In addition, Dialogic® BRI/2 Boards can facilitate four instances of DSP-based Group 3 Fax (also referred to as DSP Fax, see Note 2 below) and provide ISDN B channel data communications. Dialogic® BRI/2 Boards are currently supported only under the Windows® operating system.

Notes: ***1.*** For information on using the Dialogic® Voice API, see the *Dialogic® Voice API Programming Guide* and *Dialogic® Voice API Library Reference*.

2. For information on using DSP Fax with Dialogic® BRI/2 Boards, see the *Dialogic® Fax Software Reference*.

The Dialogic® BRI/SC and BRI/2 Boards provide network access via the ISDN Basic Rate Interface (BRI). The Dialogic® BRI/SC Boards can also function as a digital station interface, enabling direct access to BRI station sets (telephones) from PC-based computer telephony (CT) systems, and eliminating the need for local switch integration.

The Dialogic® BRI/SC Boards may also be used for connecting voice processing applications to PBX or Public Switched Telephone Network (PSTN) BRI access lines.

5.1.2 Features of BRI

BRI offers advantages or access to features not available on PRI. For example, many ISDN PBX Primary Rate products are designed as terminal equipment (TE) for connection to the central office, and cannot provide network-side access to other terminal equipment. The Dialogic® BRI/SC or BRI/2 Board can be used to connect to a PBX.

Both the Dialogic® BRI/SC and the BRI/2 Boards provide access to ISDN Layer 3 Supplemental Services. These services can be divided into two categories:

Hold and Retrieve

: Allows the application to place calls on hold, to retrieve held calls, and to respond to requests to hold or retrieve held calls using the following Global Call functions: **gc_HoldCall( )**, **gc_RetrieveCall( )**, **gc_HoldAck( )**, **gc_HoldRej( )**, **gc_RetrieveAck( )**, and **gc_RetrieveRej( )**. Refer to the function descriptions in Section 8.2, “Dialogic® Global Call Function Variances for ISDN”, on page 215 for more information.

Messaging

: Allows the application to access other supplemental services, such as Called/Calling Party Identification, Message Waiting, and Call Transfer. The services are invoked by formatting information elements (IEs) and sending them as non-call related Facility Messages (SndMsg_Facility) to the PBX or network. See the **gc_SndMsg( )** and **gc_SetInfoElem( )** functions for information on sending Facility Messages. See the **gc_GetCallInfo( )** function for information on retrieving Facility Messages. Also refer to Section 12.3, “BRI Supplemental Services”, on page 308.

Dialogic® BRI/2 Board Features

In addition to the features described above, Dialogic® BRI/2 Boards provide the following fax and data communications features:

Fax features

: Dialogic® BRI/2 Boards support Global Call DSP-based Group 3 Fax. Key features of DSP Fax include:

- Four channels of voice and fax per board
- Maximum of 16 fax channels per system (four BRI/2 Boards in one system)
- Software-based fax modem
- Compatibility with ITU-T Group 3 (T.4, T.30), ETSI NET/30

Note: For more information on using DSP Fax with Dialogic® BRI/2 Boards, see the *Dialogic® Fax Software Reference*.

Data features

Dialogic® BRI/2 Boards provide link layer access, across the B channel, which allows for reliable transfer of data across an ISDN network. The Dialogic® BRI/2 Boards offer Network Device Interface Specification (NDIS) compatibility. NDIS is a Microsoft® standard that allows for multiple network adapters and multiple protocols to coexist. NDIS permits the high-level protocol components to be independent of the Dialogic® BRI/2 Board by providing a standard interface. This means that the Dialogic® BRI/2 Board may be used by applications that use the standard networking APIs that are part of the Windows® operating system. NDIS supports the following:

- Remote Access Service (RAS) - RAS is enabled via NDIS and allows users to interact with the service selections provided by the specified dial-up networking setup.
- Point-to-Point Protocol (PPP) - PPP is a method of exchanging data packets between two computers. PPP can carry different network layer protocols over the same link. When the PPP connection sequence is successfully completed, the remote client and RAS server can begin to transfer data using any supported protocol. PPP Multilink provides the ability to aggregate two or more physical connections to form one larger logical connection, improving bandwidth and throughput for remote connections.

Dialogic® BRI/SC Board Features

The Dialogic® BRI/SC Boards provide a different set of ISDN features. Advantages and features specific to Dialogic® BRI/SC Boards include the following:

Data Link Layer Access

The Dialogic® BRI/SC Boards have data link layer access (also known as LAPD Layer 2). This feature provides for the reliable transfer of data across the physical link (physically connected devices), and sends blocks of frames with the necessary synchronization, error control, and flow control. Layer 2 access is particularly useful if you want to use an ISDN board to connect to a switch using a Layer 3 protocol that is not provided in the firmware.

Point-to-Multipoint Configuration

This feature allows BRI/SC protocols to support multiple TEs to be connected to a line that is configured to be a network. Up to eight TEs may be connected with a maximum of two active, non-held calls at a time. An unlimited number of calls may exist in a held state, but these calls cannot be retrieved if both B channels are already in use by other calls.

Tone Generation

This feature allows BRI/SC protocols, under a network configuration, to generate and play tones on any B channel with the use of the on-board DSP chip. These tones can be requested and configured by the application, or they can be generated by the firmware.

Note: Global Call provides some tone management capabilities for specific technologies. See Section 4.1.9, “Play a User-Defined Tone”, on page 121, Section 4.1.14, “Stop Currently Playing Tone (BRI Only)”, on page 133, and Section 4.1.15, “Redefine Call Progress Tone Attributes (BRI Only)”, on page 134. The Dialogic® ISDN call control library functions **cc_ToneRedefine( )**, **cc_PlayTone( )**, and **cc_StopTone( )** can also be used in this context. However, the use of the Dialogic® ISDN call control library is not officially supported and the *Dialogic® ISDN Software Reference*, in which these functions are documented, may not be included in the documentation for future Dialogic® System Releases.

Multiple D Channel Configuration

This feature allows the D channel of each line to be configured at any time, and as many times as needed. The application can configure and reconfigure the protocol for each station interface, allowing you to run different protocols on different stations simultaneously. The application can also change between User side and Network side, assign and change the Service Profile Identifier (SPID), and change other attributes such as the generation of in-band tones.

5ESS Custom Messaging

The 5ESS protocol has a custom messaging feature, which allows the application to send requests to drop calls and to redirect the state of calls. This feature is implemented using the **gc_SndMsg( )** function. See Section 8.2.38, "gc_SndMsg( ) Variances for ISDN", on page 245 for more information.

5.1.3 Typical BRI Applications

ISDN BRI technology offers call handling features, such as Automatic Call Distribution (ACD), call monitoring, and caller ID, that can be used to develop BRI applications such as the following:

- Call center and business communication platforms
- Automated call rerouting applications such as debit card services, international callback, and long distance resale
- Wireless gateway access
- Voice processing system access for the station side of ISDN PBXs
- Protocol conversion equipment, which allows the application to convert calls from one network protocol to another network protocol, without resource boards

5.2 Primary Rate Interface

The Dialogic® Global Call ISDN Primary Rate Interface (PRI) firmware supports both T1 and E1 protocols.

The T1 protocol implementations comply with the North American standard ISDN PRI and the INS-1500 standard used in Japan. In North America and Japan, the ISDN Primary Rate includes 23 voice/data channels (B channels) and one signaling channel (D channel).

The E1 protocol implementations comply with the E1 ISDN PRI protocols. The E1 ISDN Primary Rate includes 30 voice/data channels (B channels) and two additional channels: one signaling channel (D channel) and one framing channel to handle synchronization.

5.3 Using ISDN Protocols with Dialogic® DM3 Boards

ISDN protocols for Dialogic® DM3 Boards are described under the following topics:

- Configuring an ISDN Protocol
- Selecting an ISDN Protocol

5.3.1 Configuring an ISDN Protocol

When using Dialogic® DM3 Boards, protocol parameters such as bearer capability, Q.931 timers, NFAS parameters, initial channel state, inter call delay, disconnect timeout, called number type, called number plan, etc., are configurable in the CONFIG file.

Note: Changes to the CONFIG file for a particular protocol are automatically updated in the corresponding FCD file when the firmware is downloaded to the board. For more information, see the configuration guide for Dialogic® DM3 products.

5.3.2 Selecting an ISDN Protocol

When using Dialogic® DM3 Boards, select an ISDN protocol by choosing the appropriate Feature Configuration Description (FCD) file at board configuration time. The process for each supported operating system is as follows:

Linux
: Select the FCD file to use for each board in the System Configuration Description (SCD) file.

Windows®
: Select the FCD file to use for each board in the Dialogic® Configuration Manager (DCM).

See the configuration guide for Dialogic® DM3 products provided with the Dialogic® Software Release for more information.

5.4 Using ISDN Protocols with Dialogic® Springware Boards

Dialogic® Global Call API support for ISDN protocols on Dialogic® Springware Boards is described in the following topics:

- Available ISDN Protocols
- User Configurable ISDN Parameters
- Protocol Components
- Selecting an ISDN Protocol
- Using Non-Facility Associated Signaling (NFAS)

5.4.1 Available ISDN Protocols

When using Dialogic® Springware Boards, a standard ISDN interface providing 23 (T1) or 30 (E1) voice or data channels (B channels) and one signaling channel (D channel) is available. For a list of supported protocols, check the release information for your Dialogic® Software Release.

Each protocol is contained in a separate, modular binary file that can be installed and used as needed. This modular design simplifies adapting applications for use in numerous countries. User selectable options allow customization of the country dependent parameters to fit a particular application or configuration within a country (for example, switches within the same country may use the same protocol but may require different parameter values for local use). These parameters, such as trunk framing, trunk protocol type, D channel enable, inverted D channel data, and layer 2 access enable, are specified in the PRM file and may be modified at configuration time (that is, at any time before starting your application).

Note: Only one protocol (or network emulation test protocol) may be downloaded to a board at a time.

5.4.2 User Configurable ISDN Parameters

When using Dialogic® Springware Boards, the parameters listed in Table 26 may be configured by the user by modifying the ISDN parameter (*.prm*) file. The parameter values should be set in accordance with your protocol and carrier requirements. See the Release Documentation included with each protocol for details.

Table 26. Modifiable Protocol Parameters for Dialogic® Springware Boards

Parameter	Value (hex)	Description
000F	00† 01	Digital E1 trunk framing format: • 00 = G.703 framing without CRC4 (default) • 01 = G.703 framing with CRC4 Must be set to carrier requirement. For T1 applications, this parameter must be commented out (not used).
0013	00† 01	Digital trunk protocol type: • 00 = Standard T1/E1 (default) • 01 = PRI ISDN Must be set to 01 for ISDN application.
0014	00† 01	Digital T1 trunk framing format: • 00 = D4 framing (default) • 01 = ESF (Extended Super Frame) framing ESF framing is only supported in SCbus mode. For E1 applications, this parameter must be commented out (not used).
0016	00† 01 02	Enable D channel flag: • 00 = Undefined (default) • 01 = Enable D channel • 02 = Disable D channel Must be set to 01 for the board carrying the D channel and to 02 for all other boards in NFAS group or in a clear channel application.
† = Parameter file default selection; see most probable parameter defaults below.		

Table 26. Modifiable Protocol Parameters for Dialogic® Springware Boards (Continued)

Parameter	Value (hex)	Description
0023	00† 01	Inverted D channel flag: • 00 = D channel data is not inverted (default) • 01 = D channel is inverted Must be set to 01 for D channel inversion.
0024	00† 01 02 04 08 10 20 40 80	Feature flag: • 00 = ISDN layer 2 access inactive (disabled) (default). When layer 2 access is required, set to 01. • 01 = ISDN layer 2 access active (enabled) • 02 = Enable double call feature • 04 = Not used • 08 = Enable overlap sending feature • 10 = Enable host controlled release • 20 = Not used • 40 = Not used • 80 = Not used
† = Parameter file default selection; see most probable parameter defaults below.		

Each ISDN protocol parameter (*.prm*) file uses the most probable parameters for that protocol as the default setting. See Table 27 and Table 28 for a summary of the default parameter settings for each protocol.

Table 27. T1 ISDN Protocol Parameter Defaults When Using Dialogic® Springware Boards

Parameter	4ESS, 5ESS	DMS/100, DMS/250	NTT (INS1500)
000F	----	----	----
0013	01	01	01
0014	Check with carrier.		
0016	For a D channel board, set to 01; for an NFAS application; for a non D channel board, set to 02.		
0023	When using D4 framing, D channel inversion is recommended, set to 01 (also check with carrier); otherwise, ignore.		
0024	Select from list of features for parameter 0024 listed in Table 26.		

Table 28. E1 ISDN Protocol Parameter Defaults When Using Dialogic® Springware Boards

Parameter	1TR6	CTR4	DASS2	DPNSS	VN3	TPH
000F	Check with carrier.					
0013	01	01	01	01	01	01
0014	----	----	----	----	----	----
0016	01	01	01	01	01	01
0023	----	----	----	----	----	----
0024	Select from list of features for parameter 0024 listed in Table 26.					

Note: When using Non-Facility Associated Signaling (NFAS), the *nfas.cfg* file (in the */usr/dialogic/cfg* directory in Linux or the *<install_directory>\dialogic\cfg* directory in Windows®) that identifies which boards are using NFAS, must also be edited. See Section 5.4.5, "Using Non-Facility Associated Signaling (NFAS)", on page 195 and the instructions in the *nfas.cfg* file.

5.4.3 Protocol Components

When using Dialogic® Springware Boards, the following files are included for each protocol:

firmware (.FWL) file
: Contains protocol state engine as part of the protocol downloadable firmware.

firmware parameter file(s) for the protocol
: Have a file extension PRM and are located in the following directory:
  - the */usr/dialogic/data* directory in Linux
  - the *Program Files\Dialogic\data* directory in Windows®

When using Dialogic® Springware Boards, each protocol requires specific firmware parameter file(s) to be downloaded to the network boards.

5.4.4 Selecting an ISDN Protocol

When using Dialogic® Springware Boards, select the ISDN protocol to use by ensuring that the protocol firmware file and parameter file are specified in the configuration file:

Linux
: These protocol files are specified using the **ISDNProtocol** and **ParameterFile** parameters in the *dialogic.cfg* configuration file. You select the ISDN protocol to be used by selecting the **ISDNProtocol** keyword. By default, the keyword selects the protocol to be downloaded to the board and the corresponding parameter file. To specify a different parameter file, use the **ParameterFile** keyword. The following example specifies the CTR4 E1 ISDN protocol:

```
ISDNProtocol = ctr4
ParameterFile = isdnE1.prm
```

Windows®
: Use the Dialogic® Configuration Manager (DCM) to select the protocol for each board.

5.4.5 Using Non-Facility Associated Signaling (NFAS)

Non-Facility Associated Signaling (NFAS) can be set up by editing configuration files. The configuration files that must be edited are:

- *nfas.cfg* file
- ISDN parameter (*.prm*) file
- *dialogic.cfg* file

Note: No changes to an application are required. The application just needs to know that there is an additional bearer channel on the spans that are no longer using a D channel.

Editing the nfas.cfg File

Edit the *nfas.cfg* file to set up the NFAS group associations. This *nfas.cfg* file is used to inform the device driver which T1 spans are associated with which ISDN D channels. There are comments in the file that explain how to set up this file. The following is an example *nfas.cfg* file:

```
# NFAS group 1

# Board ID      Interface ID    D-Channel board ID     NFAS group ID
         1                 1                     3                 1
         2                 2                     3                 1
         3                 0                     3                 1
```

Editing the ISDN Parameter (.prm) File

Edit the ISDN parameter (*.prm*) file to disable the D channel on the spans that will be sharing the NFAS D channel. To do this, start with a properly configured D channel equipped parameter file (for example, *5ess.prm*). Make a copy of that file, naming it such that it is obvious that the two files relate but are different (for example, *5ess_NoD.prm*). In the new file, change parameter 0x0016 to value 0x02 (to disable the D channel) as indicated in the following example:

```
;---
;--- ENABLE/DISABLE the D channel (Parameter type 16H)
;--- Used only when the protocol type (Parameter number 13H) is PRI ISDN
;--- for NFAS configuration.
;--- Possible values for the data are as follows:
;--- 00H = Undefined.
;--- 01H = Enable the D channel.
;--- 02H = Disable the D channel.
0016 02
```

Editing the dialogic.cfg File

The *dialogic.cfg* file needs to edited so that the appropriate parameter file is assigned to each span. This is achieved be adding a 'ParameterFile=' line to each span in the NFAS group. The span that carries the actual NFAS D channel is assigned the base parameter file (for example, *5ess.prm*), and the spans that are sharing the NFAS D channel are assigned the modified parameter file (for example, *5ess_NoD.prm*) as indicated by the following segment from the *dialogic.cfg* file.

```
[Genload - ID 0]
ISDNProtocol=5ess
ParameterFile=5ess_NoD.prm

[Genload - ID 1]
ISDNProtocol=5ess
ParameterFile=5ess_NoD.prm

[Genload - ID 2]
ISDNProtocol=5ess
ParameterFile=5ess.prm
```

Note: The NFAS specific changes are now complete and take effect the next time the Dialogic services are started.

Building Dialogic® Global Call ISDN Applications 6

This chapter describes the Integrated Services Digital Network (ISDN)-specific header files and libraries required when building applications. Topics include:

6.1 Header Files

When compiling Dialogic® Global Call applications for the ISDN technology, it is necessary to include the following header files in addition to the standard Global Call header files, which are listed in the *Dialogic® Global Call API Library Reference* and *Dialogic® Global Call API Programming Guide*:

gcisdn.h
: ISDN-specific type definitions

Note: The *gcisdn.h* file is only required when the application uses ISDN symbols.

dm3cc_parm.h
: ISDN-specific type definitions when using Dialogic® DM3 Boards

Note: The *dm3cc_parm.h* file is only required when the application uses Dialogic® DM3 specific symbols.

6.2 Required Libraries

When building Dialogic® Global Call applications for ISDN technology, it is not necessary to link any libraries other than the standard Global Call library, *libgc.lib.*

6.3 Required System Software

The Dialogic® System Software must be installed on the development system. See the Dialogic® Software Installation Guide for your Dialogic® Software Release for more information.

Debugging Dialogic® Global Call ISDN Applications 7

This chapter describes the Dialogic® diagnostic utilities used to test and debug Integrated Services Digital Network (ISDN) applications by focusing on the connection between the application and the ISDN network. These utilities can help provide a better understanding of the effects of various ISDN configuration options on the application. Topics include:

7.1 Overview of Debugging Utilities

The ISDN diagnostic utilities:

- Aid in understanding the characteristics of an ISDN network trunk in real-time
- Reduce the need for a live network trunk or international service connections
- Simplify troubleshooting a connection

The diagnostic utilities included with the Dialogic® System Software comprise:

ISDN Network Firmware (NT1 and NE1)
: to provide network side emulation

ISDN Diagnostic Program (*isdiag*)
: to initiate calls, alter call setup parameters, and initiate traces of the D channel activity on the trunk

ISDTRACE Utility (*isdtrace*)
: to convert the binary trace files (*filename.log*) of the D channel communications into a formatted text file (*filename.res*) that can be read and analyzed. The binary trace file is generated using the **gc_StartTrace( )** and **gc_StopTrace( )** functions.

7.2 ISDN Network Firmware

Note: When performing back-to-back testing using Dialogic® DM3 Boards, the symmetric protocol option, which allows User to User or Network to Network protocol configurations, is **not** supported. The “Symmetrical C. E. Protocol” option (parameter 0x13 in the CONFIG file) must be set to 0 (disabled) for all ISDN protocols. One side should be configured to run User-side firmware (parameter 0x17 set to 0) and the other side should be configured to run Network-side firmware (parameter 0x17 set to 1).

For testing ISDN applications, the diagnostic utilities include ISDN Network Firmware to emulate the ISDN network interface. This emulator can be used to set up an ISDN PRI link between two Dialogic® network products in different host PCs so that an application can be tested without a live network connection. An ISDN PRI cable is used to connect the two network products in the two host PCs. The application runs on one host PC using the ISDN protocol specific firmware, while the other host PC runs the network emulation firmware and the ISDN diagnostic utilities.

For Dialogic® Springware Boards, use the ISDN Network Firmware as follows:

1. Connect a crossover ISDN PRI cable between the two network boards in the two host PCs.

2. Load your application in one of the host PCs.

3. Set up the other host PC to emulate the network side of the ISDN trunk by:
   - Changing the name of the protocol file and parameter file in the standard Dialogic® configuration file for the network board used to emulate the ISDN network. For Windows® applications, use the Dialogic® Configuration Manager (DCM) to make these protocol changes.
   - Setting the configurable ISDN network parameters to match those used in your application; see Table 26, “Modifiable Protocol Parameters for Dialogic® Springware Boards”, on page 192.
   - Resetting this host PC to load the emulation firmware and the ISDN network emulation parameters.

4. Run your application.

7.3 ISDN Diagnostic Program

Note: The ISDN Diagnostic program (*isdiag*) is **not** supported when using Dialogic® DM3 Boards.

When using Dialogic® Springware Boards, the ISDN Diagnostic program (*isdiag*) is an interactive tool used to help verify ISDN line operation and to assist in troubleshooting the network trunk. When your application is ready for final installation, running this diagnostic program can help in determining what the network carrier is expecting first.

With the ISDN Diagnostic program running, a trace on the inbound call will detect what the network sent. A trace on a failed outgoing call will show the cause of the failure.

When the ISDN Diagnostic Program is first started, users identify the specific board, channel number (time slot), bus type (SCbus), and board type (T1 or E1) on which outgoing calls will be made. Incoming calls may be received on any time slot. For a Linux application, you can use the F1 key to bring up the help screens and for a description of the menu items.

To run the diagnostic utilities:

1. Enter:

   ```
   isdiag parm1 parm2 parm3 parm4
   ```

   where:

   parm1
      is the board number

   parm2
      is the channel time slot number

   parm3
      is the interface type (t for T1 and e for E1)

   parm4
      is the bus type (S for SCbus)

2. After the channel number and bus mode are selected, the program automatically configures the system and displays the first level menu.

3. Select from the following actions:
   - set outbound call parameters
   - request calling party number (ANI)
   - send maintenance request
   - display information (called party subaddress, user-to-user information, B and D channel status)
   - drop call
   - make outbound call
   - play and record 24K voice files
   - stop play/record
   - set and get ISDN information elements
   - send message
   - start/stop/browse trace files
   - restart ISDN line devices and set up to receive an inbound ISDN call
   - change the current ISDN line device number
   - shell to Linux [or shell to DOS (Windows®)]
   - hold/retrieve calls (DPNSS and QSIG protocols only)
   - set supplementary DPNSS/QSIG services (intrusion, local diversion, remote diversion, virtual calls for inbound/outbound) (DPNSS and QSIG protocols only)
   - ESC exit
   - F1 - help menu; describes the main menu options

7.4 ISDTRACE Utility

Note: The ISDTRACE utility is supported in Windows® systems only. See Section 7.5, “pritrace Utility”, on page 204, for information on a trace utility that works in Linux systems.

The ISDTRACE utility analyzes the binary trace files from the ISDN Diagnostics Program. When the utility is started with the ISDTRACE command, the utility translates the binary data into a text file. The converted text file identifies the commands issued, network responses, and binary values, as well as a description of those values.

To start the ISDTRACE utility (*isdtrace*), enter:

```
isdtrace infilename.log [<outfilename>] -p
```

where:

infilename.log
: is the saved binary file generated by the **gc_StartTrace( )** function

<outfilename>
: is the ASCII text readable trace of the D channel

-p
: elects Primary Rate (PRI)

The ISDTRACE (*isdtrace*) utility creates a temporary file called *isdtemp.log*. The *isdtemp.log* file contains the hex information of the binary input file. The following shows an example of a file fragment with the translated data:

```
NET5

RECEIVE
Response=0   SAPI=0x00
TEI=0x00
0x01 0x09  Receive Ready

                                              TRANSMIT
                                              Command=0    SAPI=0x00
                                              TEI=0x00
                                                   0x01 0x0b  Receive Ready

                                              TRANSMIT
                                              Response=1   SAPI=0x00
                                              TEI=0x00
                                              0x08 0x0a  Information
                                              Dest=0   CR=0x0002
                                              SETUP(0x05)
                                          1:          SENDING COMPLETE(0xa1)
                                          1:          BEARER CAPABILITY(0x04)
                                          2:          IE Length(0x02)
                                          3:  1------- Extension Bit
                                              -00----- Coding Standard
                                              ---00000 Info. Transfer Cap.
                                          4:  1------- Extension Bit
                                              -00----- Transfer Mode
                                              ---10000 Info. Transfer Rate
                                          1:          CHANNEL ID(0x18)
                                          2:          IE Length(0x03)
                                          3:  1------- Extension Bit
```

```
                                        -0------ Interface ID Present
                                        --1----- Interface Type
                                        ---0---- Spare
                                        ----1--- Preferred/Exclusive
                                        -----0-- D-Channel Indicator
                                        ------01 Info. Channel Sel.
                                  3.2:  1------- Extension Bit
                                        -00----- Coding Standard
                                        ---0---- Number Map
                                        ----0011 Channel/Map Element
                                    4:  1------- Extension Bit
                                        -0000010 Channel Number/Slot Map
                                    1:           CALLED PARTY NUM(0x70)
                                    2:           IE Length(0x0b)
                                    3:  1------- Extension Bit
                                        -010---- Type of Number
                                        ----0001 Numbering plan ID
                                     2019933000  Number Digit(s)
                                    1:           CALLED PARTY SUBADD(0x71)
                                    2:           IE Length(0x04)
                                    3:  1------- Extension Bit
                                        -000---- Type of Subaddress
                                            0x01 Subaddress Info.
                                            0x02 Subaddress Info.
                                            0x03 Subaddress Info.
                                    1:           USER-USER(0x7e)
                                    2:           IE Length(0x4)
                                    3:      0x04 Protocol Discrim.
                                            0x44 User Information
                                            0x69 User Information
                                                   0x61 User Information

RECEIVE
Command=1    SAPI=0x00
TEI=0x00
0x01 0x0a  Receive Ready

RECEIVE
Response=0   SAPI=0x00
TEI=0x00
0x0a 0x0a  Information
Orig=1   CR=0x8002
CALL PROCEEDING(0x02)
  1:           CHANNEL ID(0x18)
  2:           IE Length(0x03)
  3:  1------- Extension Bit
      -0------ Interface ID Present
      --1----- Interface Type
      ---0---- Spare
      ----1--- Preferred/Exclusive
      -----0-- D-Channel Indicator
      ------01 Info. Channel Sel.
3.2:  1------- Extension Bit
      -00----- Coding Standard
      ---0---- Number Map
      ----0011 Channel/Map Element
  4:  0------- Extension Bit
      -0000010 Channel Number/Slot Map

                                          TRANSMIT
                                          Command=0    SAPI=0x00
                                          TEI=0x00
                                               0x01 0x0c  Receive Ready
```

```
RECEIVE
Response=0   SAPI=0x00
TEI=0x00
0x0c 0x0a  Information
Orig=1   CR=0x8002
CALL CONNECT(0x07)
                                              TRANSMIT
                                              Command=0    SAPI=0x00
                                              TEI=0x00
                                                   0x01 0x0e  Receive Ready

                                              TRANSMIT
                                              Response=1   SAPI=0x00
                                              TEI=0x00
                                              0x0a 0x0e  Information
                                              Dest=0   CR=0x0002
                                                   CALL CONNECT ACKNOWLEDGE(0x0f)
RECEIVE
Command=1    SAPI=0x00
TEI=0x00
0x01 0x0c  Receive Ready
```

7.5 pritrace Utility

Note: The *pritrace* utility is supported on Linux systems running applications that use ISDN PRI on Dialogic® DM3 and Springware Boards. On Windows® systems, the equivalent functionality is provided by the ISDTRACE utility. See Section 7.4, "ISDTRACE Utility", on page 202 for more information.

The *pritrace* utility analyzes the binary trace files generated by using the **gc_StartTrace( )** and **gc_StopTrace( )** functions. When the utility is started with the *pritrace* command, the utility translates the binary data into a text file. The converted text file identifies the commands issued, network responses, and binary values, as well as a description of those values.

To start the *pritrace* utility, enter:

```
./pritrace myfile.log <outfilename>
```

where:

./pritrace
: is the command

myfile.log
: is the saved binary file generated by the **gc_StartTrace( )** function

<outfilename>
: is the ASCII text readable trace of the D channel. If no file is specified, the *pritrace* utility generates a readable text file called *myfile.res*.

7.6 Debugging Tools When Using Dialogic® DM3 Boards

The primary tool available when debugging Dialogic® Global Call API applications that use Dialogic® DM3 boards is *isdntrace.exe*. This tool provides a trace of the ISDN messages received and transmitted with timestamps.

7.7 ISDN Trace Capability on Multiple Trunks

The capture of ISDN D channel trace information can be dynamically started and stopped via Dialogic® Global Call APIs, and logs can be collected on two or more trunks at the same time. In earlier Dialogic® System Release Software, the only available tool for collecting ISDN trace information (isdntrace) could not be run on more than one trunk. This trace information allows developers to determine the root cause of protocol issues in a system that uses Dialogic® DMT160TEC or DMN160TEC Digital Telephony Interface Boards (no other boards support this feature).

Note: Enabling ISDN tracing on a higher number of trunks causes the call performance to be severely degraded and must not be left permanently enabled in a production environment.

Trace data is presented to the application via the GCEV_TRACEDATA event, and the application is responsible for the retrieval, processing, and logging of the traced data from the event. Traced data events are generated for all signaling frames on the D channel on both inbound and outbound calls. Tracing is started by using the **gc_StartTrace( )** function after the ISDN firmware downloads. Tracing is stopped using the **gc_StopTrace( )** function.

A GCEV_TRACEDATA event is received asynchronously on a board device when a Layer 2 (LAP_D) INFORMATION frame is sent or received by the firmware. Table 29 shows the structure of the data, listing the fields and size of each field.

Table 29. Structure of GCEV_TRACEDATA Data for ISDN

Field	Size of Field
Send/Receive Flag	4
Timestamp	4
Payload	N

The following are descriptions of the fields listed in Table 29:

Send/Receive Flag
: When this field has a value of 01, it indicates a frame sent by the firmware to the network. When this field has a value of 02, it indicates a frame received by the firmware from the network.

Timestamp
: This field is a 4 byte field (unsigned 32 bit integer) representing the number of firmware timer ticks since the board was started. Each tick is equivalent to 4 milliseconds. This field is encoded in the Little Endian format (the least significant byte of the number is stored in the

lowest memory address, and the most significant byte of the number is stored in the highest memory address).

Payload

This field is a number of bytes representing the event that occurred in the firmware.

ISDN-Specific Function Information 8

This chapter describes the Dialogic® Global Call API functions that have additional functionality or perform differently when used with ISDN technology. The function descriptions are presented alphabetically and contain information that is specific to ISDN applications. Generic function description information (that is, information that is not technology-specific) is provided in the *Dialogic® Global Call API Library Reference*.

Topics in this chapter include:

8.1 Dialogic® Global Call Functions Supported by ISDN

The following is a list of all functions in the Dialogic® Global Call API library. The description under each function indicates whether the function is supported, not supported, supported with variances, or supported differently for Dialogic® Springware Boards and Dialogic® DM3 Boards.

gc_AcceptCall( )
: Supported with variances described in Section 8.2.1, "gc_AcceptCall( ) Variances for ISDN", on page 215.

gc_AcceptInitTransfer( )
: Not supported.

gc_AcceptModifyCall( )
: Not supported.

gc_AcceptXfer( )
: Not supported.

gc_AlarmName( )
: Supported.

gc_AlarmNumber( )
: Supported.

gc_AlarmNumberToName( )
: Supported.

gc_AlarmSourceObjectID( )
: Supported.

gc_AlarmSourceObjectIDToName( )
: Supported.

gc_AlarmSourceObjectName( )
Supported.

gc_AlarmSourceObjectNameToID( )
Supported.

gc_AnswerCall( )
Supported with variances described in Section 8.2.2, “gc_AnswerCall( ) Variances for ISDN”, on page 215.

gc_Attach( ) (deprecated)
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_AttachResource( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_BlindTransfer( )
Not supported.

gc_CallAck( )
Supported with variances described in Section 8.2.3, “gc_CallAck( ) Variances for ISDN”, on page 216.

gc_CallProgress( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.4, “gc_CallProgress( ) Variances for ISDN”, on page 217. For Dialogic® DM3 Boards: Not supported.

gc_CCLibIDToName( )
Supported.

gc_CCLibNameToID( )
Supported.

gc_CCLibStatus( ) (deprecated)
Supported.

gc_CCLibStatusAll( ) (deprecated)
Supported.

gc_CCLibStatusEx( )
Supported.

gc_Close( )
Supported.

gc_CompleteTransfer( )
Not supported.

gc_CRN2LineDev( )
Supported.

gc_Detach( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_DropCall( )
Supported with variances described in Section 8.2.5, “gc_DropCall( ) Variances for ISDN”, on page 218.

gc_ErrorInfo( )
Supported.

gc_ErrorValue( ) (deprecated)
Supported.

gc_Extension( )
Supported with variances described in Section 8.2.6, "gc_Extension( ) Variances for ISDN", on page 220.

gc_GetAlarmConfiguration( )
Supported.

gc_GetAlarmFlow( )
Supported.

gc_GetAlarmParm( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Not supported.

gc_GetAlarmSourceObjectList( )
Supported.

gc_GetAlarmSourceObjectNetworkID( )
Supported.

gc_GetANI( ) (deprecated)
Supported with variances described in Section 8.2.7, "gc_GetANI( ) Variances for ISDN", on page 220.

gc_GetBilling( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.8, "gc_GetBilling( ) Variances for ISDN", on page 220. For Dialogic® DM3 Boards: Not supported.

gc_GetCallInfo( )
Supported with variances described in Section 8.2.9, "gc_GetCallInfo( ) Variances for ISDN", on page 220.

gc_GetCallProgressParm( )
Not supported.

gc_GetCallState( )
Supported.

gc_GetConfigData( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.10, "gc_GetConfigData( ) Variances for ISDN", on page 221. For Dialogic® DM3 Boards: Not supported.

gc_GetCRN( )
Supported.

gc_GetCTInfo( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_GetDNIS( ) (deprecated)
Supported with variances described in Section 8.2.11, "gc_GetDNIS( ) Variances for ISDN", on page 221.

gc_GetFrame( ) (deprecated)
Supported.

gc_GetInfoElem( ) (deprecated)
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Not supported.

gc_GetLineDev( )
Supported.

gc_GetLinedevState( )
Supported.

gc_GetMetaEvent( )
Supported.

gc_GetMetaEventEx( ) (Windows® extended asynchronous model only)
Supported.

gc_GetNetCRV( ) (deprecated)
Supported.

gc_GetNetworkH( ) (deprecated)
Supported.

gc_GetParm( )
Supported with variances described in Section 8.2.12, "gc_GetParm( ) Variances for ISDN", on page 222.

gc_GetResourceH( )
Supported.

gc_GetSigInfo( )
Supported with variances described in Section 8.2.13, "gc_GetSigInfo( ) Variances for ISDN", on page 222.

gc_GetUserInfo( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.14, "gc_GetUserInfo( ) Variances for ISDN", on page 222. For Dialogic® DM3 Boards: Not supported.

gc_GetUsrAttr( )
Supported.

gc_GetVer( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Not supported.

gc_GetVoiceH( ) (deprecated)
Supported.

gc_GetXmitSlot( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_HoldACK( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.15, "gc_HoldACK( ) Variances for ISDN", on page 223. For Dialogic® DM3 Boards: Supported.

gc_HoldCall( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.16, "gc_HoldCall( ) Variances for ISDN", on page 224. For Dialogic® DM3 Boards: Supported.

gc_HoldRej( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.17, "gc_HoldRej( ) Variances for ISDN", on page 224. For Dialogic® DM3 Boards: Supported.

gc_InitXfer( )
Not supported.

gc_InvokeXfer( )
Not supported.

gc_LinedevToCCLIBID( )
Supported.

gc_Listen( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_LoadDxParm( )
Not supported.

gc_MakeCall( )
Supported with variances described in Section 8.2.18, "gc_MakeCall( ) Variances for ISDN", on page 224.

gc_Open( ) (deprecated)
Supported.

gc_OpenEx( )
Supported with variances described in Section 8.2.19, "gc_OpenEx( ) Variances for ISDN", on page 230.

gc_QueryConfigData( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Not supported.

gc_RejectInitXfer( )
Not supported.

gc_RejectModifyCall( )
Not supported.

gc_RejectXfer( )
Not supported.

gc_ReleaseCall( ) (deprecated)
Supported.

gc_ReleaseCallEx( )
Supported with variances described in Section 8.2.20, "gc_ReleaseCallEx( ) Variances for ISDN", on page 233.

gc_ReqANI( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.21, "gc_ReqANI( ) Variances for ISDN", on page 233. For Dialogic® DM3 Boards: Not supported.

gc_ReqModifyCall( )
Not supported.

gc_ReqMoreInfo( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Supported with variances described in Section 8.2.22, "gc_ReqMoreInfo( ) Variances for ISDN", on page 234.

gc_ReqService( )
Not supported.

gc_ResetLineDev( )
Supported with variances described in Section 8.2.23, "gc_ResetLineDev( ) Variances for ISDN", on page 234.

gc_RespService( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.24, "gc_RespService( ) Variances for ISDN", on page 235. For Dialogic® DM3 Boards: Not supported.

gc_ResultInfo( )
Supported.

gc_ResultMsg( ) (deprecated)
Supported.

gc_ResultValue( ) (deprecated)
Supported.

gc_RetrieveAck( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.25, "gc_RetrieveAck( ) Variances for ISDN", on page 235. For Dialogic® DM3 Boards: Supported.

gc_RetrieveCall( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.26, "gc_RetrieveCall( ) Variances for ISDN", on page 235. For Dialogic® DM3 Boards: Supported.

gc_RetrieveRej( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.27, "gc_RetrieveRej( ) Variances for ISDN", on page 235. For Dialogic® DM3 Boards: Supported.

gc_SendMoreInfo( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Supported with variances described in Section 8.2.28, "gc_SendMoreInfo( ) Variances for ISDN", on page 235.

gc_SetAlarmConfiguration( )
Supported.

gc_SetAlarmFlow( )
Supported.

gc_SetAlarmNotifyAll( )
Supported.

gc_SetAlarmParm( )
For Dialogic® Springware Boards: Supported. For Dialogic® DM3 Boards: Not supported.

gc_SetAuthenticationInfo( )
Not supported.

gc_SetBilling( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.29, "gc_SetBilling( ) Variances for ISDN", on page 236. For Dialogic® DM3 Boards: Not supported.

gc_SetCallingNum( ) (deprecated)
Supported with variances described in Section 8.2.30, "gc_SetCallingNum( ) Variances for ISDN", on page 237.

gc_SetCallProgressParm( )
Not supported.

gc_SetChanState( )
Supported with variances described in Section 8.2.31, "gc_SetChanState( ) Variances for ISDN", on page 237.

gc_SetConfigData( )
Supported with variances described in Section 8.2.32, "gc_SetConfigData( ) Variances for ISDN", on page 238.

gc_SetEvtMsk( ) (deprecated)
Supported with variances described in Section 8.2.33, "gc_SetEvtMsk( ) Variances for ISDN", on page 239.

gc_SetInfoElem( ) (deprecated for Dialogic® Springware Boards)
Supported with variances described in Section 8.2.34, "gc_SetInfoElem( ) Variances for ISDN", on page 240.

gc_SetParm( )
Supported with variances described in Section 8.2.35, "gc_SetParm( ) Variances for ISDN", on page 241.

gc_SetupTransfer( )
Not supported.

gc_SetUserInfo( )
For Dialogic® Springware Boards: Supported with variances described in Section 8.2.36, "gc_SetUserInfo( ) Variances for ISDN", on page 244. For Dialogic® DM3 Boards: Not supported.

gc_SetUsrAttr( )
Supported.

gc_SipAck( )
Not supported.

gc_SndFrame( ) (deprecated)
Supported with variances described in Section 8.2.37, "gc_SndFrame( ) Variances for ISDN", on page 245.

gc_SndMsg( ) (deprecated)
Supported with variances described in Section 8.2.38, "gc_SndMsg( ) Variances for ISDN", on page 245.

gc_Start( )
Supported.

gc_StartTrace( )
Supported with variances described in Section 8.2.39, "gc_StartTrace( ) Variances for ISDN", on page 246.

gc_Stop( )
Supported.

gc_StopTrace( )
Supported with variances described in Section 8.2.40, "gc_StopTrace( ) Variances for ISDN", on page 247.

gc_StopTransmitAlarms( )
Supported.

gc_SwapHold( )
Not supported.

gc_TransmitAlarms( )
Supported.

gc_UnListen( )
For Dialogic® Springware Boards: Not supported. For Dialogic® DM3 Boards: Supported.

gc_util_copy_parm_blk( )
Supported.

gc_util_delete_parm_blk( )
Supported.

gc_util_find_parm( )
Supported.

gc_util_find_parm_ex( )
Supported.

gc_util_insert_parm_ref( )
Supported.

gc_util_insert_parm_ref_ex( )
Supported.

gc_util_insert_parm_val( )
Supported.

gc_util_next_parm( )
Supported.

gc_util_next_parm_ex( )
Supported.

gc_WaitCall( )
Supported with variances described in Section 8.2.41, "gc_WaitCall( ) Variances for ISDN", on page 247.

8.2 Dialogic® Global Call Function Variances for ISDN

The Dialogic® Global Call API function variances that apply when using ISDN technology are described in the following sections. See the *Dialogic® Global Call API Library Reference* for generic (technology-independent) descriptions of the Global Call functions.

8.2.1 gc_AcceptCall( ) Variances for ISDN

The **gc_AcceptCall( )** function sends an Alerting message to the network to indicate that the phone is ringing and to stop the network from sending any further information. The **gc_AcceptCall( )** function can be called at the following times:

- In asynchronous mode, the function can be called any time after a GCEV_OFFERED or a GCEV_PROGRESSING event is received.
- In synchronous mode, the function can be called any time after the successful completion of a **gc_WaitCall( )** function.

This message stops the ISDN protocol timers (such as, T302, T303, T304, T310). If the application cannot answer the call within the protocol time-out value (10 seconds), then this function must be issued to stop the protocol layer 3 timer.

Dialogic® DM3 Board-specific variances

In the case of a DISCONNECT collision, if the inbound call is disconnected while the application was trying to accept the call, the application receives a GCEV_DISCONNECTED event (no GCEV_TASKFAIL event is received). When using Dialogic® DM3 Boards, the GCEV_DISCONNECTED event is a valid termination event for the **gc_AcceptCall( )** function.

Dialogic® Springware Board-specific variances

In the case of a DISCONNECT collision, if the inbound call is disconnected while the application was trying to accept the call, depending on the timing, the application may receive a GCEV_TASKFAIL event with the error code 0x10f (BADSTATE). The application should restart the time slot using the **gc_ResetLineDev( )** to handle this glare condition.

8.2.2 gc_AnswerCall( ) Variances for ISDN

The **gc_AnswerCall( )** function must be used to complete the call establishment process. The **gc_AnswerCall( )** function can be called at the following times:

- In asynchronous mode, the function can be called any time after a GCEV_OFFERED, GCEV_ACCEPT, or GCEV_PROGRESSING event is received.
- In synchronous mode, the function can be called any time after the successful completion of a **gc_WaitCall( )** function.

This function sends a Connect message to the network to indicate that the call was accepted.

Dialogic® DM3 Board-specific variances

In the case of a DISCONNECT collision, if the inbound call is disconnected while the application was trying to answer the call, the application receives a GCEV_DISCONNECTED event (no GCEV_TASKFAIL event is received). When using Dialogic® DM3 Boards, the GCEV_DISCONNECTED event is a valid termination event for the **gc_AnswerCall( )** function.

Dialogic® Springware Board-specific variances

In the case of a DISCONNECT collision, if the inbound call is disconnected while the application was trying to answer the call, depending on the timing, the application may receive a GCEV_TASKFAIL event with the error code 0x10f (BADSTATE). The application should restart the time slot by issuing a **gc_DropCall( )** followed by a **gc_ReleaseCallEx( )** to handle this glare condition.

8.2.3 gc_CallAck( ) Variances for ISDN

The **gc_CallAck( )** function allows the application to either:

- Send the first response to an incoming call after the GCEV_OFFERED event is received in asynchronous mode or after the **gc_WaitCall( )** function returns in synchronous mode. See Section 8.2.3.1, "Sending First Response to an Incoming Call", on page 216.
- Request additional DNIS (DDI) digits from the network. See Section 8.2.3.2, "Requesting Additional DNIS Digits", on page 217.

Note: B channel negotiation is not currently available.

8.2.3.1 Sending First Response to an Incoming Call

The **gc_CallAck( )** function can be used if the application needs to control the sending of the Setup acknowledge or call Proceeding message to the network, that is, the first response to the incoming call after a GCEV_OFFERED event is received. The type field in the GC_CALLACK_BLK in this context is GCACK_SERVICE_ISDN.

Most applications allow the firmware to handle the first response and therefore this feature is optional. If this feature is required, parameters in the GC_CALLACK_BLK can be set up to handle the following conditions:

- The received setup message contains insufficient destination information. The GC_CALLACK_BLK data structure can be initialized as follows:

```
callack.type = GCACK_SERVICE_ISDN;
callack.service.isdn.acceptance = CALL_SETUP_ACK;
callack.service.isdn.linedev = 0;
```

- The received setup message contains all the information necessary to set up the call. The GC_CALLACK_BLK data structure can be initialized as follows:

```
callack.type = GCACK_SERVICE_ISDN;
callack.service.isdn.acceptance = CALL_PROCEEDING;
callack.service.isdn.linedev = 0;
```

Dialogic® DM3 Board-specific variances

When using Dialogic® DM3 Boards, by default, the application controls the sending of the SETUP ACK and CALL PROCEEDING messages. The **gc_SetEvtMask( )** function can be used to change the default so that the firmware automatically sends the SETUP ACK and CALL PROCEEDING messages. See Section 8.2.33, "gc_SetEvtMsk( ) Variances for ISDN", on page 239 for more information.

Dialogic® Springware Board-specific variances

When using Dialogic® Springware Boards, by default, the SETUP ACK and CALL PROCEEDING messages are automatically sent by the firmware. The **gc_SetConfigData( )** function can be used to change the default so that the application controls the sending of the SETUP ACK and CALL PROCEEDING messages (using the GCMSK_SETUP_ACK and GCMSK_PROC_SEND bitmask parameters). See Section 4.2.20, "Set ISDN-Specific Event Masks", on page 149 for more information.

8.2.3.2 Requesting Additional DNIS Digits

The **gc_CallAck( )** function can be used to request additional DNIS information from the network. The type field in the GC_CALLACK_BLK in this context is GCACK_SERVICE_INFO.

Note: The GCACK_SERVICE_INFO define deprecates the GCACK_SERVICE_DNIS define used in previous releases.

When the digits are collected, the **gc_CallAck( )** function completes. These digits may be retrieved using the **gc_GetCallInfo( )** function with the **info_id** parameter set to DESTINATION_ADDRESS.

The following example shows how to request one additional destination address (DNIS) digit:

```
GC_CALLACK_BLK callack;
callack.type = GCACK_SERVICE_INFO;
callack.service.info.info_type = DESTINATION_ADDRESS;
callack.service.info.info_len = 1;   /* One additional digit */
```

When a GCEV_MOREINFO event is received as a termination event to **gc_CallAck( )**, the result value for the event will indicate if more digits can be retrieved. See the *Dialogic® Global Call API Library Reference* for more information.

Dialogic® DM3 Board-specific variances

When using Dialogic® DM3 Boards, the only info.info_type value supported is DESTINATION_ADDRESS.

8.2.4 gc_CallProgress( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_CallProgress( )** function is not supported when using Dialogic® DM3 Boards.

The **gc_CallProgress( )** function is obsolete. The **gc_Extension( )** function with an **ext_id** of GCIS_EXID_CALLPROGRESS is the recommended equivalent.

8.2.5 gc_DropCall( ) Variances for ISDN

In an ISDN environment, the **gc_DropCall( )** function supports all of the cause values listed in the *Dialogic® Global Call API Library Reference* with the exception of GC_SEND_SIT. In addition, the cause values listed below may be used:

ACCESS_INFO_DISCARDED
: Access information discarded

BAD_INFO_ELEM
: Information element nonexistent or not implemented

BEAR_CAP_NOT_AVAIL
: Bearer channel capability not available

CAP_NOT_IMPLEMENTED
: Bearer channel capability not implemented

CHAN_DOES_NOT_EXIST
: Channel does not exist

CHAN_NOT_IMPLEMENTED
: Channel type not implemented

FACILITY_NOT_IMPLEMENT
: Requested facility not implemented

FACILITY_NOT_SUBSCRIBED
: Facility not subscribed

FACILITY_REJECTED
: Facility rejected

GC_USER_BUSY
: End user is busy

INCOMING_CALL_BARRED
: Incoming call barred

INCOMPATIBLE_DEST
: Incompatible destination

INTERWORKING_UNSPEC
: Interworking unspecified

INVALID_CALL_REF
: Invalid call reference

INVALID_ELEM_CONTENTS
: Invalid information element

INVALID_MSG_UNSPEC
: Invalid message, unspecified

INVALID_NUMBER_FORMAT
: Invalid number format

MANDATORY_IE_LEN_ERR
: Message received with mandatory information element of incorrect length

MANDATORY_IE_MISSING
: Mandatory information element missing

NETWORK_OUT_OF_ORDER
: Network out of order

NO_CIRCUIT_AVAILABLE
: No circuit available

NO_ROUTE
: No route. Network has no route to the specified transient network or to the destination.

NO_USER_RESPONDING
: No user responding

NONEXISTENT_MSG
: Message type nonexistent or not implemented

NUMBER_CHANGED
: Number changed

OUTGOING_CALL_BARRED
: Outgoing call barred

PRE_EMPTED
: Call preempted

PROTOCOL_ERROR
: Protocol error, unspecified

RESP_TO_STAT_ENQ
: Response to status inquiry

SERVICE_NOT_AVAIL
: Service not available

TEMPORARY_FAILURE
: Temporary failure

TIMER_EXPIRY
: Recovery on timer expired

UNSPECIFIED_CAUSE
: Unspecified cause

WRONG_MESSAGE
: Message type invalid in call state or not implemented

WRONG_MSG_FOR_STATE
: Message type not compatible with call state

The **gc_DropCall( )** function sends a Disconnect message to the network to indicate that the call was terminated.

Note: A GCEV_OFFERED event may be generated after **gc_DropCall( )** is issued and before the call is released. The event would be generated on a different CRN. The application must allow for this possibility and be able to handle the event.

8.2.6 gc_Extension( ) Variances for ISDN

The **gc_Extension( )** function is provided as a generic Global Call interface to allow applications to easily access and use technology-specific features. The function provides a common unified Global Call API for technology-unique features that formerly required the support of the lower-level call control library APIs.

The **ext_id** parameter of the **gc_Extension( )** function specifies the particular extension function of the call control library to be executed. The ISDN call control library has multiple extension IDs defined. For details on each extension ID, refer to Section 2.7, “ISDN-Specific Extension IDs”, on page 39.

8.2.7 gc_GetANI( ) Variances for ISDN

The **gc_GetANI( )** function retrieves ANI information (caller ID) received in the ISDN setup message. This function assumes that the caller’s number is contained in the incoming setup message. The **gc_GetANI( )** function may be issued after a GCEV_OFFERED event or following the completion of a **gc_WaitCall( )** function.

8.2.8 gc_GetBilling( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_GetBilling( )** function is **not** supported when using Dialogic® DM3 Boards; use the **gc_GetSigInfo( )** function to retrieve billing information.

The **gc_GetBilling( )** function retrieves the “Advice of Charge” Information Element (IE) from the incoming ISDN message. This function is only valid when used for the NTT (INS1500) protocol. E1 based CTR4 service providers provide billing information that cannot be retrieved using the **gc_GetBilling( )** function; use the **gc_GetSigInfo( )** function with the **info_id** parameter set to U_IES instead.

8.2.9 gc_GetCallInfo( ) Variances for ISDN

The **gc_GetCallInfo( )** function gets the Information Elements (IEs) from the incoming ISDN message. Note that every incoming ISDN message generates an event. This function must be used immediately after the event is received if the application wants the call information. The library does not queue the call information.

The IE_BLK data structure should be used to retrieve unprocessed IEs. See the IE_BLK structure reference page in this publication for more information.

Note: For the UUI (User-to-User Information) parameter, these messages are held until retrieved by the **gc_GetCallInfo( )** function. For all other message types, the current message is overwritten when a new message is received from the network.

The User-to-User Information (UUI) data returned is application dependent. The user information return format for UUI is defined in the USRINFO_ELEM data structure. See the USRINFO_ELEM structure reference page in this publication for more information. Use the USRINFO_ELEM structure to retrieve the UUI. Ensure that the size of the information buffer is large enough to hold the UUI string.

When using the DPNSS protocol, see Section 12.2, “DPNSS IEs and Message Types”, on page 301 and Section 3.2, “DPNSS-Specific Call Scenarios”, on page 96 for more information.

Dialogic® DM3 Board-specific variances

Since the **gc_GetCallInfo( )** function does not queue information, it is recommended to use the **gc_GetSigInfo( )** function.

ISDN billing information (AOC) cannot be retrieved using the **gc_GetCallInfo( )** function. Use the **gc_GetSigInfo( )** function with the **info_id** parameter set to U_IES instead. U_IES retrieves all public IEs (as defined by Telenetworks).

8.2.10 gc_GetConfigData( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_GetConfigData( )** function is **not** supported when using Dialogic® DM3 Boards.

The **gc_GetConfigData( )** function supports the Real Time Configuration Management (RTCM) feature. The **gc_GetConfigData( )** function retrieves configuration parameter data for a given target object. A target object is a configurable basic entity and is represented by its target type and target ID. The target type identifies the kind of physical entity (e.g., time slot) with the kind of the software module (e.g., CCLib) that maintains the physical entity’s configuration data. The target ID identifies the specific target object (e.g., line device ID), which is generated by Global Call at runtime. Refer to the *Dialogic® Global Call API Library Reference* for details on this API. For the ISDN parameters that are retrievable using this API, refer to Section 4.2, “Operations Performed Using RTCM”, on page 137.

8.2.11 gc_GetDNIS( ) Variances for ISDN

The **gc_GetDNIS( )** function can be called multiple times prior to issuing a **gc_AcceptCall( )** or **gc_AnswerCall( )** function. For example, this function can be called after a GCEV_OFFERED event is received and again after a **gc_CallAck( )** function terminates.

8.2.12 gc_GetParm( ) Variances for ISDN

See Table 34, "Call Setup Parameters When Using gc_SetParm( )", on page 242 for the ISDN parameters that may be retrieved by the **gc_GetParm( )** function.

The **gc_GetParm( )** function, when used in this context, returns the information set using the **gc_SetParm( )** function. See Section 8.2.35, "gc_SetParm( ) Variances for ISDN", on page 241 for more information on the meaning of these parameters.

Dialogic® DM3 Board-specific variances

The following parameters are supported:

- GCPR_MINDIGITS
- GCPR_CALLINGPARTY
- GCPR_MEDIADETECT
- GCPR_CALLPROGRESS

8.2.13 gc_GetSigInfo( ) Variances for ISDN

The **gc_GetSigInfo( )** function gets the signaling information from an incoming ISDN message. To use the **gc_GetSigInfo( )** function for a channel, the application needs to specify the size of the queue (circular buffer, maintained internally by the call control library) by calling the **gc_SetParm( )** function and setting the RECEIVE_INFO_BUF to the desired size. Failure to set the size of RECEIVE_INFO_BUF will result in an error.

Note: The **gc_GetCallInfo( )** function can also be used to get the Information Elements (IE) from an incoming ISDN message. However, when using **gc_GetCallInfo( )**, there is only one buffer to store message information. Since it is possible to get several ISDN messages before the application has the chance to process them, it is recommended to use the **gc_GetSigInfo( )** function to retrieve and store multiple messages.

The User-to-User Information (UUI) data returned is application dependent. The user information return format for UUI is defined in the USRINFO_ELEM data structure. See the USRINFO_ELEM structure reference page in this publication for more information. Use the USRINFO_ELEM structure to retrieve the UUI. Ensure that the size of the information buffer is large enough to hold the UUI string.

When using the DPNSS protocol, see Section 12.2, "DPNSS IEs and Message Types", on page 301 and Section 3.2, "DPNSS-Specific Call Scenarios", on page 96.

8.2.14 gc_GetUserInfo( ) Variances for ISDN

Notes: ***1.*** The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_GetUserInfo( )** function is **not** supported when using Dialogic® DM3 Boards.

2. This function is **not** supported by the Dialogic® BRI/2 Board.

The **gc_GetUserInfo( )** function gets unprocessed information elements in CCITT format. The **gc_GetUserInfo( )** function must be used immediately after the message is received if the application requires the call information. The library will not queue the call information; subsequent messages on the same line device will be discarded if the previous messages are not retrieved.

Note: Since the **gc_GetUserInfo( )** function does not queue any information, use of this function is not recommended.

The following table provides the parameter inputs for the **gc_GetUserInfo( )** function.

Parameter	Input
target_type	One of the following: • GCTGT_GCLIB_NETIF • GCTGT_GCLIB_CHAN • GCTGT_GCLIB_CRN
target_id	CRN or line device ID
infoparmblkp	A pointer to a GC_PARM_BLK with the following: • Parameter Set ID: GCIS_SET_IE • Parameter ID: GCIS_PARM_UIEDATA

The following code is an example of how to use the **gc_GetUserInfo( )** function:

```
#include "gclib.h"
#include "gcerr.h"
#include "gcisdn.h"

int GetUserInfo(LINEDEV linedev)
{
   IE_BLK ie_blk;
   GC_PARM_BLK infoparm;
   int retcode;

   infoparm.pstruct = (void *)&ie_blk;
   retcode=gc_GetUserInfo(GCTGT_GCLIB_CHAN, linedev, &infoparm);

   return retcode;
}
```

Note: This function returns with an error if the Global Call application has set GCIS_PARM_RECEIVEINFOBUF. To retrieve unprocessed IEs, the application should use the **gc_Extension( )** function with the GCIS_EXID_GETSIGINFO extension ID.

8.2.15 gc_HoldACK( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_HoldACK( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, “Implementing Call Hold and Retrieve”, on page 173 for more information.

8.2.16 gc_HoldCall( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_HoldCall( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, "Implementing Call Hold and Retrieve", on page 173 for more information.

The **gc_HoldCall( )** function allows the application to place an active call on hold. For PRI protocols (including NTT, DPNSS, and QSIG) and for BRI Network-side protocols, the call must be in the Connected state to be put on hold. If the **gc_HoldCall( )** function is called prior to the Connected state, then the GCEV_HOLDREJ event will be generated. For BRI User-side, the call can be put on hold any time after the GCEV_PROCEEDING event is received.

8.2.17 gc_HoldRej( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_HoldRej( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, "Implementing Call Hold and Retrieve", on page 173 for more information.

8.2.18 gc_MakeCall( ) Variances for ISDN

The **gc_MakeCall( )** function enables the application to make an outgoing call on the specified line device. When this function is issued asynchronously, a CRN will be assigned and returned immediately if the function is successful. All subsequent communications between the application and the Global Call library regarding that call will use the CRN as a reference. If this function is issued synchronously, the CRN will be available at the successful completion of the function. The GCEV_CONNECTED event, returned after calling the **gc_MakeCall( )** function, indicates that a Connect message was received from the network.

ISDN provides the flexibility of selecting network services on any B channel. This service selection is contained in the ISDN call setup message, which may vary from network to network.

Dialogic® Springware Board-specific variances

The maximum number of digits that can be specified in the **numberstr** is dictated by the protocol switch specification. Users must know the specification limits for the protocol they are using, otherwise the protocol stack will reject the MakeCall request. If the maximum number is exceeded, a GCEV_TASKFAIL event with an unknown ISDN error will be received.

The **timeout** parameter is supported only in the synchronous mode for ISDN applications. If the **timeout** value specified expires before the remote end answers the call, then the **gc_MakeCall( )** function returns -1. The Global Call error value is set to EGC_TIMEOUT. Setting the **timeout** parameter to 0 causes this parameter to be ignored.

If using the asynchronous mode, the **timeout** parameter is ignored. The **gc_MakeCall( )** function will get a GCEV_DISCONNECTED event if the remote end does not answer before the protocol dependent timer expires. Depending on the protocol being used, the protocol dependent timer may be viewed or set using the **gc_GetConfigData( )** or **gc_SetConfigData( )** function with a set ID of GCIS_SET_DCHANCFG and a parameter ID corresponding to the protocol-specific timer. See Section 4.2.2, "Set/Retrieve Configuration of a Logical Link (BRI Only)", on page 138 for more information. For PRI protocols, the only parameters that can be set using the **gc_SetConfigData( )** function are the protocol-specific timers. For BRI protocols, many other parameters are configurable using **gc_SetConfigData( )**.

When calling the **gc_MakeCall( )** function in synchronous mode (that is, with the mode parameter set to EV_SYNC), the **timeout** parameter must be set to any value other than 0, otherwise the **gc_MakeCall( )** function will not return, causing the application to hang. Alternatively, the **gc_MakeCall( )** function can be issued in asynchronous mode (that is, with the mode parameter set to EV_ASYNC).

8.2.18.1 ISDN Setup Messages

ISDN Setup messages vary in complexity depending on the calling scenarios and the network service selections. A minimum requirement for an ISDN setup message is three mandatory call information elements (IEs): channel number (time slot number), destination number (digits), and bearer capability (characteristics of the channel). The first two elements tell the network which channel to use and the destination of the call. The third element tells the network the path for routing the call. More complex calling scenarios require the definition of additional IEs in the SETUP message.

The GC_MAKECALL_BLK associated with the **gc_MakeCall( )** function and the **gc_SetInfoElem( )** function provide developers with the ability to set call IEs for both simple and complex calling scenarios.

8.2.18.2 Using the GC_MAKECALL_BLK Structure

The GC_MAKECALL_BLK structure contains two pointers to structures that can be used to define SETUP message information.

Certain information required to fill in the GC_MAKECALL_BLK structure can only be provided by your ISDN service provider. When using the GC_MAKECALL_BLK structure, all entries must be initialized. See the GC_MAKECALL_BLK structure reference page in this publication for more information.

Note: Because ISDN services vary with switches and provisioning plans, a set of default standards cannot be set for the GC_MAKECALL_BLK structure. Therefore, it is up to the application to fill in the applicable MAKECALL_BLK values that apply to the particular provisioning.

Caution: When using **gc_MakeCall( )** to make an outbound call, if the origination.address field in the GCLIB_MAKECALL_BLK structure is set to NULL or '\0' (null string), the destination.address_plan and the destination.address_type fields in the GCLIB_MAKECALL_BLK structure are ignored. This precludes the option of using the **gc_SetCallingNum( )** function to set the origination phone number and specifying a value of NULL or '\0' for the origination_phone_number field in the GCLIB_MAKECALL_BLK structure, when the destination number plan and the destination number type values (as specified in the destination.address_plan and destination.address_type fields in the GCLIB_MAKECALL_BLK structure) must be included in the outgoing message.

Dialogic® DM3 Board-specific variances

The call setup parameters described in Table 30, "Call Setup Parameters When Using gc_MakeCall( )", on page 227 are **not** supported when using Dialogic® DM3 Boards. However, the **gc_SetInfoElem( )** function can be used to set these parameters.

Note: When using Dialogic® DM3 Boards, if both the cclib and gclib pointers in the GC_MAKECALL_BLK are set, the gclib pointer is ignored.

The following parameters in the MAKECALL_BLK are **not** supported:

- BC_xfer_mode
- usr_rate
- facility_feature_service
- facility_coding_value
- USRINFO_ELEM
- NFACILITY_ELEM
- destination_sub_number_plan
- destination_sub_phone_number
- origination_sub_number_plan
- origination_sub_phone_number

The destination_number_type and origination_number_type parameters support the following values only:

- INTL_NUMBER
- NAT_NUMBER
- EN_BLOC_NUMBER (supported by the origination_number_type parameter only)

Notes:
1. If a GC_MAKECALL_BLK structure is not specified, the default values are taken from the FCD (Feature Configuration Description) file. See the configuration information for Dialogic® DM3 products provided with the Dialogic® Software Release.
2. Neither the **gc_MakeCall( )** function nor the **gc_SetParm( )** function can be used to modify these parameters.

Dialogic® Springware Board-specific variances

When using Dialogic® Springware Boards, the following options are available:

- For simple call scenarios, you can set up information elements in gclib and set cclib to NULL.
- For more complicated call scenarios or network services, you can set up information elements in cclib and set gclib to NULL.

Table 30 shows the parameters that can be included in the GC_MAKECALL_BLK associated with the **gc_MakeCall( )** function. The default values appear in **bold**. If no default value is indicated, you need to set the parameter using the **gc_SetParm( )** function or specify the parameter in the MAKECALL_BLK data structure. Unspecified parameters that do not have default values are not included in the setup message.

Table 30. Call Setup Parameters When Using gc_MakeCall( )

Parameter	Level	Description
BC_XFER_CAP	chan	Bearer channel information transfer capacity. Possible values are: • **BEAR_CAP_SPEECH** - speech • BEAR_CAP_UNREST_DIG - unrestricted data • BEAR_CAP_REST_DIG - restricted data
BC_XFER_MODE	chan	Bearer channel information transfer mode. Possible values are: • ISDN_ITM_CIRCUIT - circuit switch
BC_XFER_RATE	chan	Bearer channel information transfer rate. Possible values are: • **BEAR_RATE_64KBPS** - 64K bps transfer rate
USRINFO_LAYER1_PROTOCOL	chan	Layer 1 protocol to use on bearer channel. Possible values are: • ISDN_UIL1_CCITTV110 - CCITT standardized rate adaptation V.110/X.30 • ISDN_UIL1_G711ULAW - Recommendation G.711 µ-Law • ISDN_UIL1_G711ALAW - Recommendation G.711 a-Law • ISDN_UIL1_CCITTV120 - CCITT standardized rate adaptation V.120 • ISDN_UIL1_CCITTX31 - CCITT standardized rate adaptation X.31 HDLC • ISDN_UIL1_G721ADCPM - Recommendation G.721 32 kbits/s ADCPM and Recommendation I.460 • ISDN_UIL1_G722G725 - Recommendation G.722 and G.725 - 7kHz audio • ISDN_UIL1_H261 - Recommendation H.261 - 384 kbits/s video • ISDN_UIL1_NONCCITT - Non-CCITT standardized rate adaptation • ISDN_UIL1_CCITTV120 - CCITT standardized rate adaptation V.120

Table 30. Call Setup Parameters When Using gc_MakeCall( ) (Continued)

Parameter	Level	Description
USRINFO_LAYER1_PROTOCOL (continued)	chan	• ISDN_UIL1_CCITTX31 - CCITT standardized rate adaptation X.31 HDLC • ISDN_UIL1_CCITTV110 - CCITT standardized rate adaptation V.110/X.30 • ISDN_UIL1_G711ULAW - Recommendation G.711 μ-Law • ISDN_UIL1_G711ALAW - Recommendation G.711 a-Law • ISDN_UIL1_G721ADCPM - Recommendation G.721 32 kbits/s ADCPM and Recommendation I.460 • ISDN_UIL1_G722G725 - Recommendation G.722 and G.725 - 7kHz audio • ISDN_UIL1_H261 - Recommendation H.261 - 384 kbits/s video • ISDN_UIL1_NONCCITT - Non-CCITT
USR_RATE †	chan	User rate to use on bearer channel (layer 1 rate). Possible values are: • ISDN_UR_EINI460 - Determined by E bits in I.460 • ISDN_UR_56000 - 56 kbits, V.6 • ISDN_UR_64000 - 64 kbits, X.1 • ISDN_UR_134 - 134.5 bits, X.1 • ISDN_UR_12000 - 12 kbits, V.6
CALLED_NUM_TYPE	chan	Called party number type. Possible values are: • **EN_BLOC_NUMBER** - number is sent en-block (in whole; not overlap sending) • INTL_NUMBER - international number for international call. Check with service provider to see if your subscription allows international calls. • NAT_NUMBER - national number for call within national numbering plan (accepted by most networks) • LOC_NUMBER - subscriber number for a local call. Check with service provider to see if your subscription allows local calls. • OVERLAP_NUMBER - overlap sending - number is not sent in whole (not available on all networks)
CALLED_NUM_PLAN	chan	Called party number plan. Possible values are: • **UNKNOWN_NUMB_PLAN** - unknown plan • ISDN_NUMB_PLAN - ISDN/telephony (E.164/E.163) (accepted by most networks) • TELEPHONY_NUMB_PLAN - telephony numbering plan • PRIVATE_NUMB_PLAN - private numbering plan

Table 30. Call Setup Parameters When Using gc_MakeCall( ) (Continued)

Parameter	Level	Description
CALLING_NUM_TYPE	chan	Calling party number type. Possible values are: • **EN_BLOC_NUMBER** - number is sent en-block (in whole - not overlap sending) • INTL_NUMBER - international number for international call. Check with service provider to see if your subscription allows international calls. • NAT_NUMBER - national number for call within national numbering plan (accepted by most networks) • LOC_NUMBER - subscriber number for a local call. Check with service provider to see if your subscription allows local calls. • OVERLAP_NUMBER - overlap sending - number is not sent in whole (not available on all networks)
CALLING_NUM_PLAN	chan	Calling party number plan. Possible values are: • **UNKNOWN_NUMB_PLAN** - unknown plan • ISDN_NUMB_PLAN - ISDN/telephony (E.164/E.163) (accepted by most networks) • TELEPHONY_NUMB_PLAN - telephony numbering plan • PRIVATE_NUMB_PLAN - private numbering plan

The **gc_SetParm( )** function can also be used to specify call setup parameters. See Section 8.2.35, "gc_SetParm( ) Variances for ISDN", on page 241 for more information.

8.2.18.3 Using the gc_SetInfoElem( ) Function

Not all optional IEs can be set using the GC_MAKECALL_BLK structure. When additional IEs are to be added to the setup message (or to other messages), then the **gc_SetInfoElem( )** function is used in combination with the GC_MAKECALL_BLK structure.

The format used in the **gc_SetInfoElem( )** function must conform to CCITT IE defined formats. The following example illustrates using the **gc_SetInfoElem( )** function in this manner.

```
#include <windows.h>             /* For Windows applications only */
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include "srllib.h"
#include "dtilib.h"
#include "gcisdn.h"

/* Global variables */

LINEDEV        lbuf;
CRN            crn_buf;
unsigned long mode = EV_ASYNC;        /* Mode = Asynchronous */

void InitIEBlk (IE_BLK *ie_blk_ptr)
{
    ie_blk_ptr->length = 6; /* 6 bytes of data */
    /* The IE header */
    ie_blk_ptr->data[0] = 0x78; /* IE type=0x78 (TRANSIT NETWORK SELECTION) */
```

```
    ie_blk_ptr->data[1] = 0x04; /* the length of IE data */
    /* The IE data */
    ie_blk_ptr->data[2] = 0xA1; /* National network & carrier ID */
    /* Carrier ID Code = 288 */
    ie_blk_ptr->data[3] = 0x32; /* 2 */
    ie_blk_ptr->data[4] = 0x38; /* 8 */
    ie_blk_ptr->data[5] = 0x38; /* 8 */
  };

void main()
{
   int rc;
   char *devname = ":N_dtiB1T1:P_isdn";
   IE_BLK ie_blk;

   /* open the device */
   if (( rc = gc_OpenEx(&lbuf, devname, EV_SYNC, NULL)) < 0)
      {
         printf("%s: ERROR %d: Unable to open\n",devname,rc);
         exit(1);
      }
   .
   .
   .
   /* Application set up 'TRANSIT NETWORK SELECTION' by using
   gc_SetInfoElem() function call.  Initialize TNS IE first.
   */

   /* setting up the TNS IE */

   InitIEBlk(&ie_blk);

   if ((rc = gc_SetInfoElem(lbuf, &ie_blk)) < 0)
      {
         printf("%s: ERROR %d: Unable to set info element\n",devname,rc);
         exit (1);
      }

   if (rc = gc_MakeCall(lbuf, &crn_buf, "1234567", NULL, 0, EV_ASYNC) == -1)
      {         printf("%s: ERROR %d: Unable to make call\n",devname,rc);
         exit (1);
      }
   .
   .
   .
}
```

The **gc_SetParm( )** function can also be used to set call setup parameters. See Section 8.2.35, "gc_SetParm( ) Variances for ISDN", on page 241 for more information.

8.2.19 gc_OpenEx( ) Variances for ISDN

The **gc_OpenEx( )** function is used to open both network board and channel (time slot) devices and optionally a voice device (Dialogic® DM3 Boards only). This generic call control function initializes the specified time slot on the specified trunk. A line device ID will be returned to the application.

From the Global Call perspective, ISDN line devices are opened in the blocked state. When a **gc_OpenEx( )** function call returns successfully, the application must wait for a GCEV_UNBLOCKED event before making or waiting for another call on the opened device. The GCEV_UNBLOCKED event indicates that the line is ready to accept calls.

The device to be opened is specified by the **devicename** parameter as defined in the **gc_OpenEx( )** function description in the *Dialogic® Global Call API Library Reference*, but there are some restrictions depending on the board architecture as described below.

Caution: If a synchronous application issues the **gc_ResetLineDev( )** function immediately after the **gc_OpenEx( )** function, all the events pending on that line device are lost. The application should wait for the GCEV_UNBLOCKED event before calling the **gc_ResetLineDev( )** function and wait for the GCEV_RESETLINEDEV event before calling any function.

Dialogic® DM3 Board-specific variances

The **gc_OpenEx( )** function supports the specification of a protocol when:

- Opening a trunk device, for example, the **devicename** string contains ":N_dtiB1..."
- Opening the *first* time slot (channel) device, for example, the **devicename** string contains ":N_dtiB1T1...", assuming that the trunk device has not already been opened

Note: Only one active protocol per trunk is allowed. All channels on a trunk must use the same protocol.

The protocol is specified in the :P_<protocol_name> part of the **devicename** string, for example "...:P_5ESS". In earlier releases, the :P_<protocol_name> part of the **devicename** string was ignored. To maintain backward compatibility, if the protocol is not included in the **devicename** string (that is, :P_<protocol_name> is not specified), the default protocol for the board, selected statically during board configuration, is used.

If the protocol for a trunk has been specified by either opening the trunk device or opening the *first* time slot (channel) device, and the application then attempts to open a time slot device on the same trunk with a different protocol, the **gc_OpenEx( )** function will fail.

The **gc_OpenEx( )** function supports the inclusion of a voice device in the **devicename** string so that a voice device is opened and implicitly attached to the network device in one **gc_OpenEx( )** call. This option is particularly useful when using Call Progress Analysis (CPA) that mandates a voice device be attached to the network device. See Section 4.15, "Call Progress Analysis When Using Dialogic® DM3 Boards", on page 172 for more information.

At the firmware level, the line is considered *blocked* until otherwise informed (that is, some event occurs to change the state). From the Global Call perspective, the line is also considered *blocked* until otherwise informed. The firmware triggers the generation of the GCEV_UNBLOCKED event. This behavior is in contrast to the behavior on Dialogic® Springware Boards as described below.

When a B channel is placed in service, a SERVICE message may be transmitted, depending on the value of the CHP SetParm 0x1312 parameter in the CONFIG file. The CHP SetParm 0x1312 parameter controls the sending of the SERVICE message when a B channel is placed in service. For more information on the CONFIG file settings, see the configuration information for Dialogic® DM3 products provided with the Dialogic® Software Release.

Dialogic® Springware Board-specific variances

The **gc_OpenEx( )** function does **not** support the specification of a protocol when opening a trunk (board) device or a time slot (channel) device. The **devicename** string must include both the protocol_name, which must be set to “isdn”, and the network_device_name fields. The following examples illustrate the devicename format when opening an ISDN PRI device:

- To open the first time slot on board dtiB3, the **devicename** format is:

```
":N_dtiB3T1:P_isdn"
```

- To open the board dtiB3, the **devicename** format is:

```
":N_dtiB3:P_isdn"
```

Each PRI structure is composed of one D channel and 23 (T1) or 30 (E1) B (bearer) channels. A PRI board device, such as dtiB1, is defined as a station and controls the D channel. A PRI time slot device, such dtiB1T1, is defined as a bearer channel under a station.

Each BRI structure is composed of one D channel and two B (bearer) channels. A BRI board device, such as briS1, is defined as a station and controls the D channel the same way as a PRI board device. A BRI time slot device, such as briS1T1, is defined as a bearer channel under a station and is handled the same way as a PRI line device.

Caution: Do not open a D or B channel more than once from the same process, or you may get unpredictable results.

The **gc_OpenEx( )** function does **not** support the inclusion of a voice device in the **devicename** string. Use the **dx_open( )** function to get a voice device handle.

At the firmware level, the line is considered *unblocked* until otherwise informed (that is, some event occurs to change the state). From the Global Call perspective, the line is considered *blocked* until otherwise informed. To reconcile this difference in behavior, the Global Call software generates the required GCEV_UNBLOCKED event as part of the **gc_OpenEx( )** functionality. This behavior is in contrast with the behavior for Dialogic® DM3 Boards as described above.

When using Dialogic® Springware Boards, if a blocking alarm exists on the line when an application tries to open a device, the **gc_OpenEx( )** function will complete, generating the GCEV_UNBLOCKED event, before the firmware detects that the alarm exists, which would trigger the generation of a GCEV_BLOCKED event. This means that the application temporarily sees a GCEV_UNBLOCKED even though an alarm exists on the line. The application must be capable of handling a GCEV_BLOCKED event at any time even milliseconds after a GCEV_UNBLOCKED event.

Caution: A multi-threaded application doing call control on Springware ISDN should have, at most, one thread per device. In other words, two or more threads should not be used to make or receive calls on a single device, such as dtiB1T1.

8.2.20 gc_ReleaseCallEx( ) Variances for ISDN

The **gc_ReleaseCallEx( )** function must be called after a **gc_DropCall( )** function terminates.

If a previous **gc_WaitCall( )** function was issued synchronously, then once the **gc_ReleaseCallEx( )** function is issued, an inbound call is either:

- routed to a different line device if channel selection is preferred
- rejected until the **gc_WaitCall( )** function is issued again

If the **gc_WaitCall( )** function is used in asynchronous mode, the inbound call notification can be received immediately after the **gc_ReleaseCallEx( )** function.

Dialogic® DM3 Board-specific variances

When using Dialogic® DM3 Boards, under PRI, the RELEASE message is controlled by the application, via the **gc_DropCall( )** function for calls disconnected by the remote side or by the **gc_ReleaseCall( )** function for calls dropped by the application.

Dialogic® Springware Board-specific variances

When using Dialogic® Springware Boards with PRI protocols, the firmware sends the RELEASE message to the network automatically, by default. However, the host can be configured to control when to send the RELEASE message to the network by using a parameter configuration file set prior to download time. Unlike PRI, the BRI board passes this control to the host application by default. The host application then sends the RELEASE message through the **gc_ReleaseCall( )** function. See Section 3.1.4, "Network-Initiated Inbound Call (Synchronous Mode)", on page 51, for more information on how to use this function.

Caution: When using Dialogic® Springware Boards with BRI protocols, under load conditions, or if the remote end delays transmitting the RELEASE COMPLETE message, an application can experience a significant delay while the **gc_ReleaseCall( )** function unblocks and returns control to the application. This delay can be up to 8 seconds long if the RELEASE COMPLETE message is never returned. The **gc_ReleaseCall( )** function is supported only in synchronous mode; therefore, this problem occurs only in applications that use the asynchronous single-threaded programming model. In this case, when this blocking function is called within a handler processing the GCEV_DROPCALL event, it could create a bottleneck for processing any other event and, thereby, could affect call-handling performance.

8.2.21 gc_ReqANI( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_ReqAni( )** function is **not** supported when using Dialogic® DM3 Boards.

The **gc_ReqANI( )** function requests the ANI information (caller ID) from ANI-on-demand networks. The ANI is usually included in the ISDN setup message. However, if the caller ID does not exist and the network provides AT&T ANI-on-demand service, the driver automatically requests the caller ID from the network if this feature is enabled.

If the ANI information is always available, use the **gc_GetANI( )** function, instead of the **gc_ReqANI( )** function, for a faster return.

Note: The **gc_ReqANI( )** function is used exclusively for the AT&T ANI-on-demand service.

The **gc_ReqANI( )** function can operate as either a multitasking or non-multitasking function. It is a multitasking function when the caller number is offered upon request and the network provides this type of service (such as AT&T ANI-on-demand service). The **gc_ReqANI( )** function is a non-multitasking function when the calling party number is received or when the network does not offer an ANI-on-demand service. Thus, if ANI is already available, the function returns immediately because it does not have to instruct the interface device to query the switch.

In EV_ASYNC mode, the function will always return an event. In EV_SYNC mode, the function will return automatically with the ANI if one is available. Otherwise, the function will wait for completion of the ANI-on-demand request.

8.2.22 gc_ReqMoreInfo( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® DM3 Boards only. The **gc_ReqMoreInfo( )** function is fully supported when using Dialogic® Springware Boards.

When a GCEV_MOREINFO event, which is a terminating event to **gc_ReqMoreInfo( )**, is received, the result value for the event indicates if more digits could be retrieved. See the *Dialogic® Global Call API Library Reference* for more information.

8.2.23 gc_ResetLineDev( ) Variances for ISDN

The **gc_ResetLineDev( )** function resets the channel to an Idle state. When this function is called, the following activities take place on the specified B channel in the order listed:

1. The active call is disconnected and all new incoming calls are blocked.
2. The CRN and all call information are cleared.
3. When the function returns, the channel will be blocked from accepting incoming calls. The application must issue a **gc_WaitCall( )** function to accept a new call.

Caution: If a synchronous application issues the **gc_ResetLineDev( )** function immediately after the **gc_OpenEx( )** function, all the events pending on that line device are lost. The application should wait for the GCEV_UNBLOCKED event before calling the **gc_ResetLineDev( )** function and wait for the GCEV_RESETLINEDEV event before calling any function.

In addition to being used after the recovery of trunk error or alarm conditions, or to reset the channel to Null state, this function may be used prior to using the **gc_Close( )** function when the application needs to exit for programming.

Note: For synchronous applications, the application must use another process to send the signal to the process controlling the line device to be disconnected. Then the controlling process can invoke the **gc_ResetLineDev( )** function and reset the line device.

When an error occurs and the **gc_ResetLineDev( )** function fails, the termination event GCEV_RESTARTFAIL is returned.

8.2.24 gc_RespService( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_RespService( )** function is **not** supported when using Dialogic® DM3 Boards.

The **gc_RespService( )** function returns a response to a requested service. The **gc_RespService( )** function is only supported for BRI protocols. See Section 4.3, "Responding to a Service Request (BRI Only)", on page 150 for more information.

8.2.25 gc_RetrieveAck( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_RetrieveAck( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, "Implementing Call Hold and Retrieve", on page 173 for more information.

8.2.26 gc_RetrieveCall( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_RetrieveCall( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, "Implementing Call Hold and Retrieve", on page 173 for more information.

8.2.27 gc_RetrieveRej( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_RetrieveRej( )** function is fully supported for all ISDN protocols when using Dialogic® DM3 Boards.

Call hold and retrieve functionality is supported on selected protocols only. See Section 4.16, "Implementing Call Hold and Retrieve", on page 173 for more information.

8.2.28 gc_SendMoreInfo( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® DM3 Boards only.

When using Dialogic® DM3 Boards, the **gc_SendMoreInfo( )** function can be used to send digits to the remote side. The **info_id** parameter in this context is set to DESTINATION_ADDRESS.

When using **gc_SendMoreInfo( )**, it is possible to specify that no more information will be sent (that is, provide a SendingComplete indication). This can be done by including in the send string (pointed to by the **info_ptr** function parameter) a period (.) character before the terminating null character.

Note: The period (.) character is recommended to provide the SendingComplete indication in most applications. However, for compatibility with SS7, the 'f' or 'F' character can be used instead of the period (.) character.

When sending overlapped digits using the **gc_SndMoreInfo( )** function, if the answering side accepts or answers the call, depending on the glare, the GCEV_SNDMOREINFO event may not be generated. The application should not wait for this event after receiving a GCEV_ALERTING, GCEV_PROCEEDING, or GCEV_CONNECTED event.

8.2.29 gc_SetBilling( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_SetBilling( )** function is **not** supported when using Dialogic® DM3 Boards.

When using Dialogic® Springware Boards, the **gc_SetBilling( )** function sets different billing rates for "900" number calls on a per call basis for networks providing the AT&T Vari-A-Bill service. In synchronous mode, this function must be used after the successful completion of either a **gc_MakeCall( )** or **gc_AnswerCall( )** function.

Note: The **gc_SetBilling( )** function is used exclusively for the AT&T Vari-A-Bill service.

For ISDN applications, the **rate_type** parameter for the **gc_SetBilling( )** function can have the following values:

ISDN_NEW_RATE
: change to a different per-minute rate

ISDN_FLAT_RATE
: change to flat charge

ISDN_FREE_CALL
: no charges call

ISDN_PREM_CHAR
: add additional charge to call

ISDN_PREM_CREDIT
: subtract charge from call

The current data structure for the **ratep** block (GC_RATE_U) is defined for AT&T only. For a description of the data structure, see the *Dialogic® Global Call API Library Reference*.

Both asynchronous (including extended asynchronous mode for Windows® applications) and synchronous modes are supported. If the **mode** parameter is set to EV_ASYNC, completion of the function is indicated by the GCEV_SETBILLING termination event.

ISDN cause values for the **gc_SetBilling( )** function are listed in Table 31. These cause values apply only to the AT&T Vari-A-Bill service.

Table 31. Cause Values for the gc_SetBilling( ) Function

Cause	Description
ISDN_FB_UNAVAIL	Flexible billing feature is not available.
ISDN_FB_BAD_OPER	Invalid operation.
ISDN_FB_BAD_ARG	Invalid argument.
ISDN_FB_RET_ERR	Return error component value.
ISDN_FB_IE_ERR	Invalid information element.
ISDN_NO_FB_INF	No flexible billing information.

Caution: This function is available only on the AT&T network and only for the PRI 4ESS protocol.

Caution: The **gc_SetBilling( )** function may not function in all service provider environments. Check whether retrieving billing information is an option with your service provider.

8.2.30 gc_SetCallingNum( ) Variances for ISDN

The **gc_SetCallingNum( )** function sets the **default** calling party number (caller ID) associated with the specific line device. When a calling party number is specified in the MAKECALL_BLK structure, then the **gc_MakeCall( )** function uses the number in the MAKECALL_BLK structure for the current call. Subsequent calls return to the default calling party number set by the **gc_SetCallingNum( )** function if the MAKECALL_BLK structure is not used. The default calling party number parameter ends with a '\0'.

Note: This function is supported by ISDN PRI software only. It is not supported for the Dialogic® BRI/2 Board.

Caution: When using **gc_MakeCall( )** to make an outbound call, if the origination.address field in the GCLIB_MAKECALL_BLK structure is set to NULL or '\0' (null string), the destination.address_plan and the destination.address_type fields in the GCLIB_MAKECALL_BLK structure are ignored. This precludes the option of using the **gc_SetCallingNum( )** function to set the origination phone number and specifying a value of NULL or '\0' for the origination_phone_number field in the GCLIB_MAKECALL_BLK structure, when the destination number plan and the destination number type values (as specified in the destination.address_plan and destination.address_type fields in the GCLIB_MAKECALL_BLK structure) must be included in the outgoing message.

8.2.31 gc_SetChanState( ) Variances for ISDN

Note: The **gc_SetChanState( )** is not supported for E1 ISDN or NTT PRI protocols, or for any BRI protocols.

When power is initially applied, all bearer channels (B channels) are placed in the In Service state.

Note: For some protocols, the D channel may need to be activated before the B channels can be placed in the In Service state.

For some protocols, when a channel is placed in an Out of Service state, all incoming and outgoing call requests are rejected.

The **gc_SetChanState( )** function can be used to place a B channel in an Out of Service state to avoid unnecessarily rejecting calls. Valid states for the **chanstate** parameter are:

- GCLS_INSERVICE (0)
- GCLS_MAINTENANCE (1)
- GCLS_OUT_OF_SERVICE (2)

Note: This feature may not be available in some countries.

Caution: E1 ISDN protocols do not support any message for putting B channels in an In Service or Out of Service state, therefore, the **gc_SetChanState( )** function cannot be used when using E1 ISDN protocols to avoid receiving incoming calls on some channels. An E1 application that is not ready to, or does not want to, receive incoming calls on some B channels should not issue the **gc_WaitCall( )** function on the respective channels, or it should issue the **gc_ResetLineDev( )** function on the respective channels to cancel the waitcall operation.

Caution: The **gc_SetChanState( )** function affects only the link between the calling process and the device. Other processes and devices are not affected.

Dialogic® DM3 Board-specific variances

When a B channel is placed in service, a SERVICE message may be transmitted, depending on the value of the CHP SetParm 0x1312 parameter in the CONFIG file. The CHP SetParm 0x1312 parameter controls the sending of the SERVICE message when a B channel is placed in service. For more information on the CONFIG file settings, see the configuration information for Dialogic® DM3 products provided with the Dialogic® Software Release.

8.2.32 gc_SetConfigData( ) Variances for ISDN

The **gc_SetConfigData( )** function supports the Global Call Real Time Configuration Management (RTCM) feature. The **gc_SetConfigData( )** function updates the configuration data for a given target object. A target object is a configurable basic entity and is represented by its target type and target ID. The target type identifies the kind of physical entity (for example, time slot) with the kind of the software module (for example, CCLib) that maintains the physical entity's configuration data. The target ID identifies the specific target object (for example, line device ID), which is generated by Global Call at runtime.

Dialogic® DM3 Board-specific variances

The **gc_SetConfigData( )** function is supported for:

- Call Progress Analysis (CPA) functionality; see Section 4.15, "Call Progress Analysis When Using Dialogic® DM3 Boards", on page 172.
- Dynamic trunk configuration; see Section 4.17, "Using Dynamic Trunk Configuration", on page 174.

Dialogic® Springware Board-specific variances

See Section 4.2, “Operations Performed Using RTCM”, on page 137 for information on the operations that can be performed using the **gc_SetConfigData( )** function.

8.2.33 gc_SetEvtMsk( ) Variances for ISDN

All event masks set by the **gc_SetEvtMsk( )** function, other than GCMSK_BLOCKED and GCMSK_UNBLOCKED, act on the entire trunk (board). If the mask is set on any time slot level device, the event mask for **all** time slot level line devices on that board will be set to the same value. Similarly, when an event mask is set to a particular value on a trunk level line device, then all time slot level devices connected to that trunk will have the same event mask value.

Dialogic® DM3 Board-specific variances

For ISDN technology, all of the masks described in the **gc_SetEvtMsk( )** function reference page in the *Dialogic® Global Call API Library Reference* are supported. Table 32 shows the variances for ISDN technology, which include one mask that has a different default value and one mask that is ISDN-specific.

Table 32. Mask Variances for Dialogic® DM3 Boards

Parameter Value	Description	Default
GCMSK_PROCEEDING †	Set mask for GCEV_PROCEEDING event	enabled
GCMSK_PROGRESS ‡	Set mask for GCEV_PROGRESSING event	enabled

† A mask that has a different default value than that described in the *Dialogic® Global Call API Library Reference*.
‡ A mask applicable to ISDN technology.

Note: Using the **gc_SetEvtMsk( )** function to disable the GCEV_BLOCKED or GCEV_UNBLOCKED events is supported, but **not** recommended.

In addition, the GCMSK_PROC_SEND mask has a special purpose. It does not enable or disable the reception of any Global Call event, but it defines how the firmware behaves when an incoming SETUP message is received as follows:

- If disabled, the firmware automatically sends CALL PROCEEDING or SETUP ACK messages.
- If enabled, the firmware does not send any messages automatically. This is the default for Dialogic® DM3 Boards.

The **gc_SetEvtMask( )** function can be used to enable or disable the GCMSK_PROC_SEND mask and therefore control the behavior of the firmware.

Dialogic® Springware Board-specific variances

For ISDN technology, all of the masks described in the **gc_SetEvtMsk( )** function reference page in the *Dialogic® Global Call API Library Reference* are supported with the exception of

GCMSK_BLOCKED and GCMSK_UNBLOCKED masks which are **not** supported. Table 33 shows other variances for ISDN technology.

Table 33. Mask Variances for Dialogic® Springware Boards

Parameter Value	Description	Default
GCMSK_NOFACILITYBUF †	Enable or disable for no facility buffer event	enabled
GCMSK_NOUSRINFO †	Enable or disable for no user information event	enabled
GCMSK_PROGRESS †	Enable or disable for sending progress event	enabled
GCMSK_SETUP_ACK †	Enable or disable for setup acknowledgement event	enabled
† A mask applicable to ISDN technology.		

In addition, the GCMSK_PROC_SEND mask has a special purpose. It does not enable or disable the reception of any Global Call events, but it defines how the firmware behaves when an incoming SETUP message is received as follows:

- If disabled, the firmware automatically sends CALL PROCEEDING or SETUP ACK messages. This is the default for Dialogic® Springware Boards.
- If enabled, the firmware does not send any messages automatically.

The **gc_SetEvtMask( )** function can be used to enable or disable the GCMSK_PROC_SEND mask and therefore control the behavior of the firmware.

8.2.34 gc_SetInfoElem( ) Variances for ISDN

The **gc_SetInfoElem( )** function allows the application to include application-specific ISDN IEs in the next outgoing message. The IE_BLK data structure is used by this function to set additional IEs. See the IE_BLK structure reference page in this publication for more information.

If IEs are to be included in an outgoing message, the **gc_SetInfoElem( )** function **must** be used immediately before calling any function that sends an ISDN message. ISDN message sending functions are:

- **gc_AcceptCall( )**
- **gc_AnswerCall( )**
- **gc_CallAck( )**
- **gc_CallProgress( )**
- **gc_DropCall( )**
- **gc_MakeCall( )**
- **gc_ReleaseCall( )**
- **gc_SndFrame( )**
- **gc_SndMsg( )**

When using the DPNSS protocol, see Section 3.2, "DPNSS-Specific Call Scenarios", on page 96 for more information.

Note: The **gc_SetInfoElem( )** can be used with any call control function except **gc_ReleaseCallEx( )**.

8.2.35 gc_SetParm( ) Variances for ISDN

Dialogic® DM3 Board-specific variances

With the exception of the GCPR_MINDIGITS and RECEIVE_INFO_BUF parameters, the parameters in Table 34 are **not** supported. However, the parameters can be set using the **gc_SetInfoElem( )** function.

When using Dialogic® DM3 Boards, the following parameters are supported:

GCPR_MINDIGITS
: The minimum number of digits to receive before a call is offered to the application.

GCPR_CALLINGPARTY
: The calling party number.

GCPR_RECEIVE_INFO_BUF
: The size of the buffers used to store signaling information in incoming ISDN messages.

GCPR_MEDIADETECT
: Used to enable or disable post-connect call progress or media detection; disabled by default.

GCPR_CALLPROGRESS
: Used to enable or disable call progress; enabled by default. In ISDN, pre-connect call analysis is not used, therefore the function of this parameter is as follows:

- If this parameter is enabled, post-connect call progress is dependent on the GCPR_MEDIADETECT parameter above.
- If this parameter is disabled, call progress is completely disabled regardless of the value of GCPR_MEDIADETECT parameter above.

Dialogic® Springware Board-specific variances

Table 34 lists the parameter selections that can be set using the **gc_SetParm( )** function when using Dialogic® Springware Boards. All valid values are defined in the *isdncmd.h* header file. Default values are shown in **bold**. If no default value is indicated, you need to set the parameter using the **gc_SetParm( )** function or specify the parameter in the MAKECALL_BLK data structure. Unspecified parameters that do not have default values are not included in the setup message.

Table 34. Call Setup Parameters When Using gc_SetParm( )

Parameter	Level	Description
BC_XFER_CAP	chan	Bearer channel information transfer capacity. Possible values are: • **BEAR_CAP_SPEECH** - speech • BEAR_CAP_UNREST_DIG - unrestricted data • BEAR_CAP_REST_DIG - restricted data
BC_XFER_MODE	chan	Bearer channel information transfer mode. Possible values are: • ISDN_ITM_CIRCUIT - circuit switch
BC_XFER_RATE	chan	Bearer channel information transfer rate. Possible values are: • **BEAR_RATE_64KBPS** - 64K bps transfer rate
USRINFO_LAYER1_ PROTOCOL	chan	Layer 1 protocol to use on bearer channel. Possible values are: • ISDN_UIL1_CCITTV110 - CCITT standardized rate adaptation V.110/X.30 • ISDN_UIL1_G711ULAW - Recommendation G.711 µ-Law • ISDN_UIL1_G711ALAW - Recommendation G.711 a-Law • ISDN_UIL1_CCITTV120 - CCITT standardized rate adaptation V.120 • ISDN_UIL1_CCITTX31 - CCITT standardized rate adaptation X.31 HDLC • ISDN_UIL1_G721ADCPM - Recommendation G.721 32 kbits/s ADCPM and Recommendation I.460 • ISDN_UIL1_G722G725 - Recommendation G.722 and G.725 - 7kHz audio • ISDN_UIL1_H261 - Recommendation H.261 - 384 kbits/s video • ISDN_UIL1_NONCCITT - Non-CCITT standardized rate adaptation
USR_RATE	chan	User rate to use on bearer channel (layer 1 rate). Possible values are: • ISDN_UR_EINI460 - Determined by E bits in I.460 • ISDN_UR_56000 - 56 kbits, V.6 • ISDN_UR_64000 - 64 kbits, X.1 • ISDN_UR_134 - 134.5 bits, X.1 • ISDN_UR_12000 - 12 kbits, V.6

Table 34. Call Setup Parameters When Using gc_SetParm( ) (Continued)

Parameter	Level	Description
CALLED_NUM_TYPE	chan	Called party number type. Possible values are: • **EN_BLOC_NUMBER** - number is sent en-block (in whole - not overlap sending) • INTL_NUMBER - international number for international call. Check with service provider to see if your subscription allows international calls. • NAT_NUMBER - national number for call within national numbering plan (accepted by most networks) • LOC_NUMBER - subscriber number for a local call. Check with service provider to see if your subscription allows local calls. • OVERLAP_NUMBER - overlap sending - number is not sent in whole (not available on all networks)
CALLED_NUM_PLAN	chan	Called party number plan. Possible values are: • **UNKNOWN_NUMB_PLAN** - unknown plan • ISDN_NUMB_PLAN - ISDN/telephony (E.164/E.163) (accepted by most networks) • TELEPHONY_NUMB_PLAN - telephony numbering plan • PRIVATE_NUMB_PLAN - private numbering plan
CALLING_NUM_TYPE	chan	Calling party number type. Possible values are: • **EN_BLOC_NUMBER** - number is sent en-block (in whole - not overlap sending) • INTL_NUMBER - international number for international call. Check with service provider to see if your subscription allows international calls. • NAT_NUMBER - national number for call within national numbering plan (accepted by most networks) • LOC_NUMBER - subscriber number for a local call. Check with service provider to see if your subscription allows local calls. • OVERLAP_NUMBER - overlap sending - number is not sent in whole (not available on all networks)
CALLING_NUM_PLAN	chan	Calling party number plan. Possible values are: • **UNKNOWN_NUMB_PLAN** - unknown plan • ISDN_NUMB_PLAN - ISDN/telephony (E.164/E.163) (accepted by most networks) • TELEPHONY_NUMB_PLAN - telephony numbering plan • PRIVATE_NUMB_PLAN - private numbering plan
CALLING_ PRESENTATION	chan	Calling presentation indicator. Possible values are: • **PRESENTATION_ALLOWED** - allows the display of the calling number at the remote end

Table 34. Call Setup Parameters When Using gc_SetParm( ) (Continued)

Parameter	Level	Description
CALLING_SCREENING	chan	Calling screening indicator field. Possible values are: • **USER_PROVIDED** - user provided, not screened (passes through)
GCPR_MINDIGITS	trunk	Sets minimum number of DDI digits to collect prior to terminating **gc_WaitCall( )**. GCPR_MINDIGITS may be set using the **gc_SetParm( )** function. This parameter value cannot be retrieved using the **gc_GetParm( )** function. Possible values are any positive value that indicates the number of digits expected before GCEV_OFFERED is received.
RECEIVE_INFO_BUF	chan	Multiple IE buffer. Sets the size, that is, the number of messages that can be stored in the information queue. The maximum size of the queue is MAX_RECEIVE_INFO_BUF. **Note:** The **gc_SetParm( )** function can be called only once in the application to set the RECEIVE_INFO_BUF buffer size. For **gc_SetParm( )**, the function returns <0 on failure, 0 on success. For **gc_GetParm( )**, the buffer number is returned. Possible values are any number in the range of 1 to MAX_RECEIVE_INFO_BUF (currently defined as 160).

8.2.36 gc_SetUserInfo( ) Variances for ISDN

Note: The variances described in this section apply when using Dialogic® Springware Boards only. The **gc_SetUserInfo**( ) function is **not** supported when using Dialogic® DM3 Boards, but the **gc_SetInfoElem**( ) function can be used instead. See Section 8.2.34, “gc_SetInfoElem( ) Variances for ISDN”, on page 240 for more information.

The **gc_SetUserInfo**( ) function is used to set additional information elements (IEs), allowing the application to include application-specific ISDN information elements in the next outbound message. This function is used for rapid deployment of an application that “interworks” with the network to take advantage of ISDN’s capabilities. A typical application is user-to-user information elements in each outgoing message.

Note: See Section 12.2, “DPNSS IEs and Message Types”, on page 301, for descriptions of ISDN IEs that are specific to the DPNSS protocol. The **duration** parameter should be set to C_SINGLECALL – The information elements specified by this function are applicable only to the next outgoing ISDN message.

Caution: This function must be used immediately before calling a function that sends an ISDN message. The information elements specified by this function are applicable only to the next outgoing ISDN message. The linedevice handle in the parameter must be same as the one used in the function call that sends the ISDN message. The IE data length must not exceed GCIS_MAXLEN_IEDATA of 254 bytes.

The following is an example:

```
#include "gclib.h"
#include "gcerr.h"
#include "gcisdn.h"

int SetUserInfo(LINEDEV linedev)
{
    char ie_blk;
    GC_PARM_BLKP infoparm = NULL;
    int retcode;

    ie_blk[0] = (char)0xa1; //Sending complete IE

    gc_util_insert_parm_ref(&infoparm, GCIS_SET_IE, GCIS_PARM_IEDATA, 1, &ie_blk);
    retcode=gc_SetUserInfo(GCTGT_GCLIB_CHAN, linedev, infoparm, GC_SINGLECALL);

    return retcode;
}
```

8.2.37 gc_SndFrame( ) Variances for ISDN

The **gc_SndFrame**( ) function sends a Layer 2 frame to the ISDN data link layer. The following ISDN L2 block structure can be passed to the function via the GC_L2_BLK structure:

```
l2_blk_ptr[0] = 0x08;      /* Protocol discriminator */
l2_blk_ptr[1] = 0x02;      /* CRN length - 2 bytes */
l2_blk_ptr[2] = 0x03;      /* CRN = 8003 */
l2_blk_ptr[3] = 0x80;
l2_blk_ptr[4] = 0x6e;      /* msg type = NOTIFY */

/* The first IE */
l2_blk_ptr[5] = 0x27;      /* IE type = 27 (NOTIFY) */
l2_blk_ptr[6] = 0x01;      /* The length of NOTIFY */
l2_blk_ptr[7] = 0xF1;      /* Notify indication */

/* The second IE */
l2_blk_ptr[8] = 0x76;      /* IE type = 76 (REDIRECTION) */
l2_blk_ptr[9] = 0x03;      /* length of redirection */
l2_blk_ptr[10] = 0x01;     /* unknown type and E164 plan */
l2_blk_ptr[11] = 0x03;     /* network provides presentation */
l2_blk_ptr[12] = 0x8D;     /* reason = transfer */
```

Note: When using Dialogic® DM3 Boards, the **gc_SndFrame( )** function returns an error if the number of bytes in the transmit frame exceeds the limit of 260 bytes.

8.2.38 gc_SndMsg( ) Variances for ISDN

The **gc_SndMsg( )** function uses a **msg_type** parameter to specify the type of message to send to the network. The values for **msg_type** are defined in the *isdnlib.h* header file. Supported message types are listed below. See also Section 12.2, "DPNSS IEs and Message Types", on page 301 and Section 3.2, "DPNSS-Specific Call Scenarios", on page 96 for protocol-specific information.

Dialogic® DM3 Board-specific variances

Note: The **gc_SndMsg( )** function is **not** deprecated when using Dialogic® DM3 Boards. The **gc_Extension( )** function (the suggested alternative when using Dialogic® Springware Boards) is not currently supported for this purpose.

The following message types are supported when using Dialogic® DM3 Boards:

- SndMsg_Congestion
- SndMsg_Facility
- SndMsg_Information
- SndMsg_Notify
- SndMsg_Progress
- SndMsg_RawEEM
- SndMsg_Status
- SndMsg_StatusEnquiry
- SndMsg_UsrInformation

Dialogic® Springware Board-specific variances

Note: The **gc_SndMsg( )** function is a deprecated function. The suggested alternative is **gc_Extension( )**.

The following message types continue to be supported when using Dialogic® Springware Boards:

- SndMsg_Congestion
- SndMsg_Divert †
- SndMsg_Facility
- SndMsg_FacilityACK
- SndMsg_FacilityREJ
- SndMsg_Information
- SndMsg_Intrude †
- SndMsg_Notify
- SndMsg_NSI †
- SndMsg_Transfer †
- SndMsg_Transit †
- SndMsg_UsrInformation

† Denotes messages specific to the DPNSS protocol.

8.2.39 gc_StartTrace( ) Variances for ISDN

The **gc_StartTrace( )** function should not be used during normal operations or when running an application for an extended period of time, since this function increases the processing load on the system and can quickly generate a large log file.

The **linedev** parameter must use the line device number for the D channel board. The resulting log file can be decoded using the *pritrace* utility.

The trace initiated by this function continues until a **gc_StopTrace( )** function is issued for the line device. The application should call the **gc_StopTrace( )** function before calling the **gc_Close( )** function for that line device.

Dialogic® DM3 Board-specific variances

When using the **gc_StartTrace( )** function, multiple boards can be traced at a time.

Dialogic® Springware Board-specific variances

When using the **gc_StartTrace( )** function, only one board can be traced at a time. An error is returned if the **gc_StartTrace( )** function is issued when a trace is currently running on another board.

8.2.40 gc_StopTrace( ) Variances for ISDN

The **gc_StopTrace( )** function discards any trace information stored in memory.

8.2.41 gc_WaitCall( ) Variances for ISDN

A B channel (time slot) is considered as *barred* at board download time or after a **gc_ResetLineDev( )** function is issued. To consider a B channel as *unbarred*, the client application has to issue a **gc_WaitCall( )** function on the corresponding line device.

An incoming call that is explicitly requesting a *barred* or *busy* B channel is automatically rejected with the appropriate cause value, while an incoming call that is **not** explicitly requesting a *barred* or *busy* B channel is automatically offered on one of the *unbarred* and Idle B channels, if any.

To make a call that is not explicitly requesting a specific B channel, the client application has to issue a **gc_SetInfoElem( )** to override the default Channel_Id_IE information element prior to issuing the **gc_MakeCall( )** function.

The GCEV_OFFERED event returned after calling the **gc_WaitCall( )** function indicates that a setup message was received from the network.

ISDN-Specific Parameter Reference 9

This chapter describes the parameter set IDs (set IDs) and parameter IDs (parm IDs) used with Integrated Services Digital Network (ISDN) technology. Topics include:

9.1 GCIS_SET_ADDRESS Parameter Set

Note: The GCIS_SET_ADDRESS parameter set is **not** supported when using Dialogic® DM3 Boards. Use **gc_MakeCall( )** to set and **gc_GetSigInfo( )** to retrieve called number and calling number information.

Table 35 shows the parameter IDs for the GCIS_SET_ADDRESS set ID.

Table 35. GCIS_SET_ADDRESS Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_CALLEDADDRESSPLAN	int	Called number plan. Valid values and GC-CC mapping are: • GCADDRPLAN_UNKNOWN – unknown number plan • GCADDRPLAN_ISDN – ISDN/telephony (E.164/E.163) (accepted by most networks) • GCADDRPLAN_TELEPHONY – telephony numbering plan • GCADDRPLAN_PRIVATE – private numbering plan
GCIS_PARM_CALLEDADDRESSTYPE	int	Called number type. Valid values and GC-CC mapping are: • GCADDRTYPE_INTL – international number for international call. (Verify availability with service provider.) • GCADDRTYPE_NAT – national number for call within national numbering plan (accepted by most networks) • GCADDRTYPE_LOC – subscriber number for a local call. (Verify availability with service provider.)
GCIS_PARM_CALLINGADDRESSPLAN	int	Calling number plan. Valid values and GC-CC mapping are: • GCADDRPLAN_UNKNOWN – unknown number plan • GCADDRPLAN_ISDN – ISDN/telephony (E.164/E.163) (accepted by most networks) • GCADDRPLAN_TELEPHONY – telephony numbering plan • GCADDRPLAN_PRIVATE – private numbering plan
GCIS_PARM_CALLINGADDRESSTYPE	int	Calling number type. Valid values and GC-CC mapping are: • GCADDRTYPE_INTL – international number for international call. (Verify availability with service provider.) • GCADDRTYPE_NAT – national number for call within national numbering plan (accepted by most networks) • GCADDRTYPE_LOC – subscriber number for a local call. (Verify availability with service provider.

9.2 GCIS_SET_BEARERCHNL Parameter Set

Note: In general, the GCIS_SET_BEARERCHNL parameter set is **not** supported when using Dialogic® DM3 Boards; the one exception is when using QSIG E1 and T1 protocols. Use **gc_MakeCall( )** to set and **gc_GetSigInfo( )** to retrieve bearer channel information transfer capability or information transfer rate. Setting the bearer channel information transfer rate is not supported, but **gc_GetSigInfo( )** can be used to retrieve bearer channel information transfer mode.

Table 36 shows the parameter IDs for the GCIS_SET_BEARERCHNL set ID.

Table 36. GCIS_SET_BEARERCHNL Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_CODINGSTANDARD (QSIG on Dialogic® DM3 Boards only)	int	Bearer channel coding standard to use. Valid values are: • ISDN_CODINGSTD_INTL • ISDN_CODINGSTD_CCITT
GCIS_PARM_TRANSFERCAP (QSIG on Dialogic® DM3 Boards only)	int	Bearer channel Information Transfer Capability. Valid values are: • BEAR_CAP_UNREST_DIG
GCIS_PARM_TRANSFERMODE	int	Bearer channel Information Transfer Mode. Valid values are: • ISDN_ITM_CIRCUIT • ISDN_ITM_PACKET **Note:** If this parameter is not present in the makecall block, then CCLIB will use the value set through RTCM.
GCIS_PARM_TRANSFERRATE	int	Bearer channel, Information Transfer Rate. Valid values are: • BEAR_RATE_64KBPS • BEAR_RATE_128KBPS • BEAR_RATE_384KBPS • BEAR_RATE_1536KBPS • BEAR_RATE_1920KBPS • PACKET_TRANSFER_MODE (QSIG on Dialogic® DM3 Boards only) **Note:** If this parameter is not present in the makecall block, then CCLIB will use the value set through RTCM.

9.3 GCIS_SET_CALLPROGRESS Parameter Set

Note: The GCIS_SET_CALLPROGRESS parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 37 shows the parameter IDs for the GCIS_SET_CALLPROGRESS set ID.

Table 37. GCIS_SET_CALLPROGRESS Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_CALLPROGRESS_INDICATOR	int	Specifies the progress indicator. Valid values are: • CALL_NOT_END_TO_END_ISDN – In drop-and-insert configurations, the application has the option of providing this information to the network. • IN_BAND_INFO – In drop-and-insert configurations, the application has the option of notifying the network that in-band tones are available.
GCIS_PARM_CALLPROGRESS_TONETYPE	unsigned char	Indicates the type of call progress tone. Valid values are: • 0x01 – Dialtone • 0x02 – Busytone • 0x03 – Reorder • 0x04 – Ringback

9.4 GCIS_SET_CALLTYPE Parameter Set

Table 38 shows the parameter IDs for the GCIS_SET_CALLTYPE set ID.

Table 38. GCIS_SET_CALLTYPE Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_CALL_TYPE	int	Specifies the call type. Valid values are: • CALLTYPE_NCAS – Identifies a call as an NCAS call. • CALLTYPE_CIRCUIT – Identifies a call as a normal circuit-switched call.

9.5 GCIS_SET_CHANSTATE Parameter Set

Note: The GCIS_SET_CHANSTATE parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 39 shows the parameter IDs for the GCIS_SET_CHANSTATE set ID.

Table 39. GCIS_SET_CHANSTATE Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_BCHANSTATE	int	This holds the status of B channel. Valid values are: • ISDN_IN_SERVICE – B channel is in service • ISDN_MAINTENANCE – B channel is in maintenance • ISDN_OUT_OF_SERVICE – B channel is out of service
GCIS_PARM_DCHANSTATE	int	This holds the status of D channel. Valid values are: • DATA_LINK_UP – Channel layer 2 is operable. The firmware will attempt to activate the logical link if it is not already activated and will allow the network side to establish the logical link if requested. • DATA_LINK_DOWN – Channel layer 2 is inoperable. The firmware will attempt to release the logical link if it is currently established. The firmware will allow the network side to establish the logical link if requested.

9.6 GCIS_SET_DCHANCFG Parameter Set

Note: The GCIS_SET_DCHANCFG parameter set is **not** supported when using Dialogic® DM3 Boards.

The parameter set is used to configure the Digital Subscriber Loop (DSL) for the D channel. Setting the configuration causes the activation of links if the switch type specified is valid. This set encapsulates the DSL-specific and logical Data Link-specific parameters. These parameters include switch type, switch side (Network or User), and terminal assignment (fixed Terminal Endpoint Identifier or auto-initializing Terminal Endpoint Identifier). Each station interface can be configured separately, which allows you to run different protocols on different stations simultaneously.

When the switch is operating as the User side in North American protocols, the **gc_SetConfigData( )** function (to set DSL configuration) is used to program the Service Profile Identifier (SPID). The SPID must be transmitted and acknowledged by the switch (see the **gc_RespService( )** function for more information).

The **gc_SetConfigData( )** function (to set DSL configuration) is also used to define Layer 3 timer values, specify Layer 2 Access, and set firmware features such as firmware-applied call progress tones.

Although the **gc_SetConfigData( )** function (to set DSL configuration) is supported for BRI/2 and PRI protocols, it can be used only to define Layer 3 timer values. All other parameters in the set are applicable only to BRI/SC.

Table 40 shows the parameter IDs for the GCIS_SET_DCHANCFG set ID.

Table 40. GCIS_SET_DCHANCFG Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_DCHANCFG_ AUTOINITFLAG	char	Boolean value defining whether the terminal is an auto initializing terminal. This field applies only when configuring the DSL as the User side and only to North American protocols. • AUTO_INIT_TERMINAL – auto initializing terminal • NON_INIT_TERMINAL – non-auto initializing term
GCIS_PARM_DCHANCFG_ FIRMWARE_FEATUREMASKA	int	Firmware feature control field A. This is a bit mask field for setting features in the firmware. The following defines are used to configure the firmware features. The lowest two bits provide a combination of four possible settings for the TONE feature. • NO_PCM_TONE – Disable firmware from providing tones and set default encoding according to switch type • ULAW_PCM_TONE – Provide tones and use ULAW encoding for B channel tones • ALAW_PCM_TONE – Provide tones and use ALAW encoding for B channel tones • DEFAULT_PCM_TONE – Provide tones and use default encoding for B channel tones according to the switch type setting • SENDING_COMPLETE_ATTACH – Add Sending Complete IE to SETUP message • USER_PERST_L2_ACT – Persistent L2 activation on User side • HOST_CONTROLLED_RELEASE – Delay RELEASE reply until host issues **gc_ReleaseCall( )**
GCIS_PARM_DCHANCFG_ FIRMWARE_FEATUREMASKB	int	Firmware feature control field. This is a bit mask field for setting features in the firmware. Currently not used.
GCIS_PARM_DCHANCFG_ FIXEDTEIVALUE	int	Defines the TEI to be used for a fixed TEI assigning terminal. Valid values lie in range 0 to 63. (Required when GCIS_PARM_DCHANCFG_TEIASSIGNMENT = FIXED_TEI_TERMINAL.)
GCIS_PARM_DCHANCFG_ L2ACCESS	int	Boolean value used to configure the DSL for direct layer 2 access or for full stack access. Valid values are: • LAYER_2_ONLY – ISDN access at layer 2. If this is selected then no other parameters are required. • FULL_ISDN_STACK – ISDN access at L3 call control.
GCIS_PARM_DCHANCFG_ NUMENDPOINTS	int	Number of logical data links to be supported. Valid values lie in the range 1 to MAX_DLINK, where MAX_DLINK is currently set to 8. This field only has significance when configuring the DSL as the NETWORK side.
GCIS_PARM_DCHANCFG_SPID	char	Defines the assigned Service Provider Identifier (SPID) value for terminal initialization. An ASCII digit string limited to the digits 0 to 9 and limited in length to MAX_SPID_SIZE. It is only applicable to User side US switches. **Note:** When you set the SPID, it is assigned to both bearer channels associated with the D channel. To subsequently modify SPID assignments, use this parameter to modify the value. MAX_SPID_SIZE = (20+1) – Required when GCIS_PARM_DCHANCFG_AUTOINITFLAG = AUTO_INIT_TERMINAL. Most North American switches require a SPID.

Table 40. GCIS_SET_DCHANCFG Parameter IDs (Continued)

Parameter ID	Type	Description
GCIS_PARM_DCHANCFG_ SWITCHSIDE	int	Boolean value defining whether the DSL should be configured as the Network side (NT) or the User side (TE). Valid values are: • USER_SIDE – User side of ISDN protocol • NETWORK_SIDE – Network side of ISDN protocol
GCIS_PARM_DCHANCFG_ SWITCHTYPE	int	Basic rate protocol (switch type) for DSL. Multiple run-time selectable switch types are available. Valid values are: • ISDN_BRI_5ESS – ATT 5ESS BRI • ISDN_BRI_DMS100 – Northern Telecom DMS100 BRI • ISDN_BRI_NTT – Japanese INS-Net 64 BRI • ISDN_BRI_NET3 – EuroISDN BRI • ISDN_BRI_NI1 – National ISDN 1 • ISDN_BRI_NI2 – National ISDN 2
GCIS_PARM_DCHANCFG_ TEIASSIGNMENT	int	Applies to User Side only. It specifies if the terminal has a fixed TEI or an auto-assigning TEI. If it is fixed, then GCIS_PARM_DCHANCFG_FIXEDTEIVALUE must be specified. Valid values are: • AUTO_TEI_TERMINAL – auto TEI assigning Term • FIXED_TEI_TERMINAL – Fixed TEI assigning Term The following timers define the Layer 3 timer values. • GCIS_PARM_DCHANCFG_TMR302 • GCIS_PARM_DCHANCFG_TMR303 • GCIS_PARM_DCHANCFG_TMR304 • GCIS_PARM_DCHANCFG_TMR305 • GCIS_PARM_DCHANCFG_TMR306 • GCIS_PARM_DCHANCFG_TMR308 • GCIS_PARM_DCHANCFG_TMR309 • GCIS_PARM_DCHANCFG_TMR310 • GCIS_PARM_DCHANCFG_TMR312 • GCIS_PARM_DCHANCFG_TMR313 • GCIS_PARM_DCHANCFG_TMR318 • GCIS_PARM_DCHANCFG_TMR319 • GCIS_PARM_DCHANCFG_TMR322 (long) Values are not needed.

9.7 GCIS_SET_DLINK Parameter Set

Table 41 shows the parameter IDs for the GCIS_SET_DLINK set ID.

Table 41. GCIS_SET_DLINK Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_DLINK_CES (supported on Dialogic® Springware Boards only)	char	The connection endpoint suffix. This is zero for PRI. The connection endpoint suffix specifies the telephone equipment associated with the station. Currently, for BRI, eight IDs (1 - 8) are supported when used as a network-side terminal. When used as a station-side terminal, only one ID (1) is supported.
GCIS_PARM_DLINK_SAPI (supported on Dialogic® Springware Boards only)	char	Service access pointer identifier. This is zero for BRI and PRI and 16 for X.25 packets over D channel.
GCIS_PARM_DLINK_STATE (supported on Dialogic® DM3 and Springware Boards)	int	Holds data link state. Valid values are: • DATA_LINK_UP – Channel layer 2 is operable. The firmware will attempt to activate the logical link if it is not already activated and will allow the network side to establish the logical link if requested. • DATA_LINK_DOWN – Channel layer 2 is inoperable. The firmware will attempt to release the logical link if it is currently established. The firmware will allow the network side to establish the logical link if requested. • DATA_LINK_DISABLED – Channel layer 2 was disabled and cannot be reestablished. The firmware will attempt to release the logical link if it is currently established. The firmware will not allow the network side to establish the logical link if requested.

9.8 GCIS_SET_DLINKCFG Parameter Set

Note: The GCIS_SET_DLINKCFG parameter set is **not** supported when using Dialogic® DM3 Boards.

The GCIS_SET_DLINKCFG set ID encapsulates parameters required to initialize the firmware structures to allow the logical link to be used.

Table 42 shows the parameter IDs for the GCIS_SET_DLINKCFG set ID.

Table 42. GCIS_SET_DLINKCFG Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_DLINKCFG_PROTOCOL	int	The protocol to be used on this logical link. For instance: • DATA_LINK_PROTOCOL_Q931 – indicates that the link is to be used as an ISDN connection-oriented logical link. • DATA_LINK_PROTOCOL_X25 – indicates that the link is to be used as an X.25 packet-switched link.
GCIS_PARM_DLINKCFG_STATE	int	The original state in which the logical link should be configured. Valid values are: • DATA_LINK_UP – Channel layer 2 is operable. The firmware will attempt to activate the logical link if it is not already activated and will allow the network side to establish the logical link if requested. • DATA_LINK_DOWN – Channel layer 2 is inoperable. The firmware will attempt to release the logical link if it is currently established. The firmware will allow the network side to establish the logical link if requested. • DATA_LINK_DISABLED – Channel layer 2 was disabled and cannot be reestablished. The firmware will attempt to release the logical link if it is currently established. The firmware will not allow the network side to establish the logical link if requested.
GCIS_PARM_DLINKCFG_TEI	char	Terminal Endpoint Identifier. Valid values are: • 0 to 63 – for manual TEIs (chosen by the user side) • AUTO_TEI – for automatic TEIs (chosen by the network side)

9.9 GCIS_SET_EVENTMSK Parameter Set

Note: The GCIS_SET_EVENTMSK parameter set is **not** supported when using Dialogic® DM3 Boards. See Section 8.2.33, "gc_SetEvtMsk( ) Variances for ISDN", on page 239 for more information on masking events on Dialogic® DM3 Boards.

Table 43 shows the parameter IDs for the GCIS_SET_EVENTMSK set ID. Some ISDN specific event masks do not have corresponding GC masks. All such masks are exposed through the parameter IDs shown.

Table 43. GCIS_SET_EVENTMSK Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_ADDMSK	int	Enables notification of events specified in the bitmask in addition to previously set events. Valid masks are specified below. • GCISMSK_STATUS – Receiving GCEV_EXTENSION when a status message is received from the network. Default: Not enabled. • GCISMSK_STATUS_ENQUIRY – Receiving GCEV_EXTENSION when a status enquiry message is received from the network. When this event arrives, the application should respond with a status message using **gc_SndMsg( )**/**gc_Extension( )**. The firmware will not auto respond to this message. Default: Not enabled. • GCISMSK_TMREXPEVENT – Receiving the GCEV_EXTENSION event when some timer expires at the firmware in Layer 3. Timer ID, Call ID, and the value of the timer are returned. Default: Not enabled. • GCMSK_ALERTING – Receiving the GCEV_EXTENSION event when a ringback tone has been received from the remote central office and the called party's line is now ringing. Default: Not enabled. • GCMSK_PROCEEDING – Receiving the GCEV_EXTENSION event when the call is sent out and enters the proceeding state. Default: Not enabled. • GCMSK_PROGRESS – Receiving the GCEV_EXTENSION event when an incoming progress message is received. Default: Not enabled. • GCMSK_SETUP_ACK – Receiving the GCEV_EXTENSION event when an incoming setup ACK message. Default: Not enabled.
GCIS_PARM_SETMSK	int	Enables notification of events specified in the bitmask and disables notification of previously set events. Valid masks are specified above for GCIS_PARM_ADDMSK.
GCIS_PARM_SUBMSK	int	Disables notification of events specified in the bitmask. Valid masks are specified above for GCIS_PARM_ADDMSK.

9.10 GCIS_SET_FACILITY Parameter Set

Note: The GCIS_SET_FACILITY parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 44 shows the parameter IDs for the GCIS_SET_FACILITY set ID. The parm IDs, GCIS_PARM_FACILITY_FEATURESERVICE and GCIS_PARM_FACILITY_CODING_VALUE, must be paired to support the specific feature or service requested from the network. The application writer needs to know what specific feature/service is being used before entering a value for these parameters.

Table 44. GCIS_SET_FACILITY Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_FACILITY_ FEATURESERVICE	unsigned char	Identifies facility request as a feature or a service. Valid values are: • ISDN_FEATURE – request is a facility feature. Features are normally used in the facility message after a call is initiated. Features can also be used in the setup message. • ISDN_SERVICE – requested facility is a service. Service can be used at any time in the NSF IE. Service is often used in the setup message to select a specific network service. **Note:** If this parameter is not present in the makecall block, ISDN_NOTUSED is put in the CC makecall block.
GCIS_PARM_FACILITY_ CODINGVALUE	unsigned char	Facility coding value. Identifies the specific feature or service provided. Valid values are: • ISDN_CPN_PREF – facility coding, CPN preferred • ISDN_BN_PREF – facility coding, BN preferred • ISDN_CPN – facility coding, CPN • ISDN_BN – facility coding, BN • ISDN_SDN – service coding, SDN • ISDN_MEGACOM800 – service coding, MEGACOM 800 • ISDN_MEGACOM – service coding, MEGACOM • ISDN_WATS – service coding, WATS • ISDN_TIE – service coding, TIE • ISDN_ACCUNET – service coding, ACCUNET SDS **Note:** If this parameter is not present in the makecall block, ISDN_NOTUSED is put in the CC makecall block.

9.11 GCIS_SET_GENERIC Parameter Set

Note: The GCIS_SET_GENERIC parameter set is **not** supported when using Dialogic® DM3 Boards. Use **gc_SetInfoElem( )** to set and **gc_GetSigInfo( )** to retrieve the calling presentation indicator, calling screening indicator, or the subaddress number. The calling multiple IE buffer size cannot be retrieved. The directory number cannot be set or retrieved.

Table 45 shows the parameter IDs for the GCIS_SET_GENERIC set ID.

Table 45. GCIS_SET_GENERIC Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_CALLINGPRESENTATION	int	Calling presentation indicator. Valid values are: • PRESENTATION_ALLOWED – allows the display of the calling number at the remote end
GCIS_PARM_CALLINGSCREENING	int	Calling screening indicator. Values are user-provided.
GCIS_PARM_CRNTYPE	int	Identifies the CRN type. Valid values are: • GLOBAL CRN – pertaining to all calls or channels on a trunk • NULL CRN – not related to any particular call
GCIS_PARM_DIRECTORYNUMBER	unsigned char array of max length 255	Directory Number (applicable to BRI User Side switches only).
GCIS_PARM_EVENTDATAP	void *	Used to pass event data buffer pointer from app to CCLIB to retrieve event specific information.
GCIS_PARM_NETCRV	int	Holds the network call reference value.
GCIS_PARM_RECEIVEINFOBUF	int	Multiple IE buffer. Sets the size of the buffer, that is, the number of messages that can be stored in the information queue. Valid values are in the range of 1 to MAX_RECEIVE_INFO_BUF. **Note:** The **gc_SetConfigData( )** function fails when attempting to set this parameter more than once. Setting this parameter more than once is **not** supported.
GCIS_PARM_SUBADDRESSNUMBER	unsigned char array of max length 255	Subaddress Number (applicable to BRI User Side switches only).

9.12 GCIS_SET_IE Parameter Set

Note: The GCIS_SET_IE parameter set is **not** supported when using Dialogic® DM3 Boards. Use **gc_SetParm( )** to set the calling multiple IE buffer size. The calling multiple IE buffer size cannot be retrieved.

Table 46 shows the parameter IDs for the GCIS_SET_IE set ID.

Table 46. GCIS_SET_IE Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_IEDATA	char array, length should not exceed GCIS_MAXLEN_ IEDATA	This parameter is used to pass information elements in the following: • GCIS_EXID_GETFRAME • GCIS_EXID_GETNONCALLMSG • GCIS_EXID_SNDFRAME • GCIS_EXID_SNDMSG • GCIS_EXID_SNDNONCALLMSG • **gc_SetUserInfo( )**
GCIS_PARM_UIEDATA	char array, length should not exceed GCIS_MAXLEN_ IEDATA	This parameter is used to pass unprocessed IEs from the call control library (CCLIB) to the application.

9.13 GCIS_SET_SERVREQ Parameter Set

Note: The GCIS_SET_SERVREQ parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 47 shows the parameter IDs for the GCIS_SET_SERVREQ set ID.

Table 47. GCIS_SET_SERVREQ Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_SERVREQ_CAUSEVALUE	unsigned char	Valid values are: • NETWORK_OUT_OF_ORDER • BAD_INFO_ELEM • INVALID_ELEM_CONTENTS • TIMER_EXPIRY • PROTOCOL_ERROR For a description of cause values, see Table 48.
GCIS_PARM_SERVREQ_INTERPRETER	unsigned char	Specifies how the usid and tid values are to be interpreted. Possible value settings are: • 0 – terminal is selected when it matches both the USID and TID • 1 – terminal is selected when it matches the USID but not the TID
GCIS_PARM_SERVREQ_TID	unsigned char	Terminal Identifier. The range is 01 to 63. 00 signifies that the firmware is to provide a default.
GCIS_PARM_SERVREQ_USID	unsigned char	User Service Identifier. The range is 01 to FF. 00 signifies default.

Table 48. GCIS_PARM_SERVREQ_CAUSEVALUE Values

Cause Value	Q.850 Description	Meaning
NETWORK_OUT_OF_ORDER	Network out of order	Used when the network has removed the TEI, causing the data link to go down.
BAD_INFO_ELEM	Information element/parameter non-existent or not implemented	Switch does not support endpoint initialization.
INVALID_ELEM_CONTENTS	Invalid information element contents	SPID was most likely coded incorrectly.
TIMER_EXPIRY	Recovery on timer expiry	Application tried two attempts at initialization with no response from the network.
FPROTOCOL_ERROR	Protocol error, unspecified	Used when no cause was given for the rejection.

9.14 GCIS_SET_SNDMSG Parameter Set

Note: The GCIS_SET_SNDMSG parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 49 shows the parameter ID for the GCIS_SET_SNDMSG set ID.

Table 49. GCIS_SET_SNDMSG Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_SNDMSGTYPE	int	Valid ISDN message types (protocol dependent) are: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus

9.15 GCIS_SET_TONE Parameter Set

Note: The GCIS_SET_TONE parameter set is **not** supported when using Dialogic® DM3 Boards.

Table 50 shows the parameter IDs for the GCIS_SET_TONE set ID.

Table 50. GCIS_SET_TONE Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_TONE_AMP1	short	Specifies the amplitude of the tone. The range is -40 to +3 dB.
GCIS_PARM_TONE_AMP2	short	Specifies the amplitude of the tone. The range is -40 to +3 dB.
GCIS_PARM_TONE_DURATION	unsigned short	Specifies the duration of the tone in 10 ms units. The range is 1 to 65535. Set to -1 to play forever.
GCIS_PARM_TONE_FREQ1	unsigned short	Specifies the frequency of the tone. The range is 200 to 3100 Hz.
GCIS_PARM_TONE_FREQ2	unsigned short	Specifies the frequency of the tone. The range is 200 to 3100 Hz.
GCIS_PARM_TONE_OFF1	unsigned short	Specifies the tone interval, in 10 ms units. The range is 0 to 65534 ms. Set to 0 to play a continuous tone.
GCIS_PARM_TONE_ON1	unsigned short	Specifies the tone interval, in 10 ms units. The range is 1 to 65535 ms. Set to 1 or greater for continuous tone.
GCIS_PARM_TONE_TERMPARMLENGTH	unsigned short	Duration for which tone has to be played.

ISDN-Specific Data Structures 10

This chapter describes the data structures that are specific to Integrated Services Digital Network (ISDN) technology.

Note: These data structures are defined in the *isdnlib.h* header file.

DCHAN_CFG

```
typedef struct {
   byte           layer2_access;          /* Layer 2 or full stack */
   byte           switch_type;            /* Layer 3 switch type */
   byte           switch_side;            /* Network or User side */
   byte           number_of_endpoints;    /* # of logical data links */
   byte           feature_controlA;       /* Firmware feature mask A */
   byte           feature_controlB;       /* Firmware feature mask B */
   byte           rfu_1;                  /* Reserved for future use */
   byte           rfu_2;                  /* Reserved for future use */
   struct {

      byte        tei_assignment;         /* Auto assignment or Fixed TEI terminal */
      byte        fixed_tei_value;        /* TEI value if Fixed TEI terminal */
      union {
         struct {
            byte  auto_init_flag;         /* Auto initializing term or not */
            byte  SPID[MAX_SPID_SIZE];    /* SPID for terminal, NULL
                                             terminated string. */
            byte  rfu_1;
            byte  rfu_2;
         } no_am; /* North America */
     } protocol_specific;
   } user;
  #define RFU_COUNT 8 /* # of reserve for future use bytes */
   byte rfu[RFU_COUNT];

   union {
      struct {
         long     T302;
         long     T303;
         long     T304;
         long     T305;
         long     T306;
         long     T308;
         long     T309;
         long     T310;
         long     T312;
         long     T322;
      } nt;
      struct {
         long     T303;
         long     T304;
         long     T305;
         long     T308;
         long     T310;
         long     T312;
         long     T313;
         long     T318;
         long     T319;
      } te;
   } tmr;
} DCHAN_CFG, *DCHAN_CFG_PTR;
```

■ Description

Note: The DCHAN_CFG data structure is **not** supported when using Dialogic® DM3 Boards.

The DCHAN_CFG data structure contains D channel configuration block information. The D channel configuration block sets the configuration of the Digital Subscriber Loop (DSL) for BRI applications.

Field Descriptions

The fields of the DCHAN_CFG data structure are described as follows:

layer2_access
: A boolean value used to configure the DSL for direct layer 2 access or for full stack access. Possible values are:

- LAYER_2_ONLY (0) – ISDN access at layer 2.

Note: If LAYER_2_ONLY is selected, no other parameters are required.

- FULL_ISDN_STACK (1) – ISDN access at L3 call control.

switch_type
: Basic rate protocol (switch type) for DSL. Multiple run-time selectable switch types are available. Possible values are:

- ISDN_BRI_5ESS – for the AT&T 5ESS BRI protocol
- ISDN_BRI_DMS100 – for the Northern Telecom DMS100 BRI protocol
- ISDN_BRI_NTT – for the Japanese INS-Net 64 BRI protocol
- ISDN_BRI_NET3 – for the EuroISDN BRI protocol
- ISDN_BRI_NI1 – for the National ISDN 1 protocol
- ISDN_BRI_NI2 – for the National ISDN 2 protocol

switch_side
: A boolean value defining whether the DSL should be configured for the Network side (NT) or the User side (TE). Possible values are:

- USER_SIDE (0) – for a user side protocol
- NETWORK_SIDE (1) – for a network side protocol

number_of_endpoints
: The number of logical data links to be supported. Possible values are in the range 1 to MAX_DLINK, where MAX_DLINK is currently set to 8. This field is only significant when configuring the DSL as the NETWORK side.

feature_controlA
: Firmware feature control field A. This is a bit mask field for setting features in the firmware. The following defines are used to configure the firmware features. The lowest two bits provide a combination of four possible settings for the TONE feature.

- NO_PCM_TONE (0x00) – Disable firmware from providing tones and set default encoding according to switch type
- ULAW_PCM_TONE (0x01) – Provide tones and use ULAW encoding for B channel tones
- ALAW_PCM_TONE (0x02) – Provide tones and use ALAW encoding for B channel tones
- DEFAULT_PCM_TONE (0x03) – Provide tones and use default encoding for B channel tones according to the switch type setting
- SENDING_COMPLETE_ATTACH (0x04) – Add Sending Complete IE to SETUP message
- USER_PERST_L2_ACT (0x08) – Persistent L2 activation on User side
- HOST_CONTROLLED_RELEASE (0x10) – Delay RELEASE reply until host issues **gc_ReleaseCall( )**

feature_controlB
: Firmware feature control field B. Currently not used.

rfu_1

Reserved for future use.

rfu_2

Reserved for future use.

tei_assignment

Applies to the User Side only. A boolean value that specifies if the terminal has a fixed TEI or an auto-assigning TEI. If the terminal has a fixed TEI, then the fixed_tei_value field must be specified (see below). Possible values are:

- AUTO_TEI_TERMINAL (0) – Auto TEI assigning terminal
- FIXED_TEI_TERMINAL (1) – Fixed TEI assigning terminal

fixed_tei_value

Defines the TEI to be used for a fixed TEI assigning terminal. Possible values are in the range 0 to 63. This parameter is required when tei_assignment is set to FIXED_TEI_TERMINAL.

auto_init_flag

A boolean value that defines whether the terminal is an auto initializing terminal. This field applies only when configuring the DSL at the User side and only to North American protocols. Possible values are:

- AUTO_INIT_TERMINAL (0) – Auto initializing terminal
- NON_INIT_TERMINAL (1) – Non-auto initializing terminal

SPID

Defines the assigned Service Provider Identifier (SPID) value for terminal initialization. Only applicable to User side North American switches. When you set the SPID, it is assigned to both bearer channels associated with the D channel. The value is a NULL terminated string consisting of ASCII digits limited to the digits 0-9 and limited in length to MAX_SPID_SIZE (20 + 1).

Note: This field is required when auto_init_flag is set to AUTO_INIT_TERMINAL. Most North American switches require a SPID.

no_am.rfu_1

Reserved for future use.

no_am.rfu_2

Reserved for future use.

rfu[RFU_COUNT]

Array of fields reserved for future use.

T3xxx (T302, T303, T304, T305, T306, T308, T309, T310, T312, T313, T318, T319, T322)

Defines the Layer 3 timer values. See the Q.931 specification and corresponding switch specifications for exact definitions and default values for these timers. Not all timers are applicable to all of the switches. Specified values are in 10 millisecond increments. For example, a specified value of 100 is equivalent to 1 second. Possible values are:

- 0 – Default value for switch
- 1 – Default value for switch
- 0 < n < 1 – Timer value in tens of milliseconds

Note: Incorrect or unreasonable timer settings will result in undesirable effects to calls as well as the call control stack. Before you override the default values, you need to understand the timer meanings and their interdependencies.

DLINK

```
typedef struct
{
    char  sapi;
    char  ces;
} DLINK, *DLINK_PTR;
```

Description

Note: The DLINK structure is **not** supported when using Dialogic® DM3 Boards.

The DLINK data structure contains information about the data link information block and is used in the following structures:

- SPID_BLK
- TERM_BLK
- TERM_NACK_BLK
- USPID_BLK

Field Descriptions

The fields of the DLINK data structure are described as follows:

sapi
: The Service Access Pointer Identifier (SAPI). This field is zero for ISDN PRI protocols.

ces
: The Connection Endpoint Suffix (CES). This field is zero for ISDN PRI protocols.

DLINK_CFG

```
typedef struct
{
    char  tei;
    int   state;
    int   protocol;
} DLINK_CFG, *DLINK_CFG_PTR;
```

Description

Note: The DLINK_CFG structure is **not** supported when using Dialogic® DM3 Boards.

The DLINK_CFG structure contains information about the data link logical link configuration block.

Field Descriptions

The fields of the DLINK_CFG data structure are described as follows:

tei
: Terminal Endpoint Identifier (TEI). Valid values are:
  - 0 to 63 – for manual TEIs (chosen by the user side)
  - AUTO_TEI – for automatic TEIs (chosen by the network side)

state
: The original state in which the logical link should be configured. Valid values are:
  - DATA_LINK_UP – the firmware will attempt to activate the logical link if it is not already activated. The firmware will allow the network side to establish the logical link if requested.
  - DATA_LINK_DOWN – the firmware will attempt to release the logical link if it is currently established. The firmware will allow the network side to establish the logical link if requested.
  - DATA_LINK_DISABLED – the firmware will attempt to release the logical link if it is currently established. The firmware will not allow the network side to establish the logical link if requested.

protocol
: The protocol to be used on this logical link. For example:
  - DATA_LINK_PROTOCOL_Q931 – indicates that the link is to be used as an ISDN connection-oriented logical link.
  - DATA_LINK_PROTOCOL_X25 – indicates that the link is to be used as an X.25 packet-switched link.

GC_MAKECALL_BLK

```
typedef struct
{
   GCLIB_MAKECALL_BLK  *gclib;
   void                *cclib;
} GC_MAKECALL_BLK, *GC_MAKECALL_BLKP;
```

■ Description

The GC_MAKECALL_BLK structure contains information used by the **gc_MakeCall( )** function when setting up a call.

The fields in the GC_MAKECALL_BLK structure point to other structures containing information elements (IEs) that are sent on the network. These IEs must conform to the switch-specific recommendations. Use the assumptions described in the following paragraphs when constructing IEs. See also Section 8.2.34, "gc_SetInfoElem( ) Variances for ISDN", on page 240.

Assumption 1

Variable length IEs must be provided in ascending order in the Public part, as shown in the following table.

IE Type	Value
Network Specific Facilities	0x20
Display	0x28
Signal	0x34

Assumption 2

A single byte IE (with the exception of a LOCKING Shift IE) can be placed anywhere in the message. This includes Type 1 (NON-LOCKING Shift) and Type 2 elements. The NON-LOCKING shift should cause the code shift in the forward direction only. For example, when in codeset "3," the NON-LOCKING shift should add an element in codeset "4." See Table 51 for Type 1 settings and Table 52 for Type 2 settings.

Table 51. NON-LOCKING Shift IEs - Type 1

IE Type	Value	Codeset
Network Specific Facilities	0x20	0
Shift	0x9E	6 (NON-LOCKING)
IPU	0x76	6
Display	0x28	0
Signal	0x34	0

Table 52. Single Byte IEs - Type 2

IE Type	Value	Codeset
Network Specific Facilities	0x20	0
Sending Complete	0xA1	0 (Single Byte IE)
Display	0x28	0
Signal	0x34	0

Assumption 3

A LOCKING Shift IE must be placed after all the IEs when a lower codeset is included. A NON-LOCKING Shift IE or another LOCKING Shift IE of a greater codeset value can follow the IE. See Table 53 and Table 54 for two options for setting LOCKING Shift IEs.

Table 53. LOCKING Shift IEs - Option 1

IE Type	Value	Codeset
Network Specific Facilities	0x20	0
Sending Complete	0xA1	0 (Single Byte IE)
Display	0x28	0
Signal	0x34	0
Shift	0x94	4 (LOCKING)
IPU	0x76	4
Shift	0x9E	6 (NON-LOCKING)
DDD	0x55	6
SSS	0x44	4
Shift	0x97	7 (LOCKING)
ABC	0x77	7
DEF	0x77	7

Table 54. LOCKING Shift IEs - Option 2

IE Type	Value	Codeset
Network Specific Facilities	0x20	0
Sending Complete	0xA1	0 (Single Byte IE)
Display	0x28	0
Keypad Facility	0x2C	0
Shift	0x96	6 (LOCKING)
IPU	0x76	6
Shift	0x90	0 (NON-LOCKING)
Signal	0x34	0

Table 54. LOCKING Shift IEs - Option 2 (Continued)

IE Type	Value	Codeset
ABC	0x77	6
DEF	0x77	6
Shift	0x97	7 (LOCKING)
ABC	0x77	7
DEF	0x77	7

Assumption 4

User-supplied IEs (with the exception of CHANNEL_ID_IE, see below) take precedence over the Firmware-defined IEs, even those that are in the private IE parts.

Assumption 5

The CHANNEL_ID_IE will always be taken from the Firmware-defined section.

Assumption 6

When Single Byte IEs and NON-LOCKING Shift IEs occur in both the User-supplied and Firmware-defined sections, the value is taken from the User-defined section. However, this value will be inserted at the position defined by the firmware when the firmware has a specific requirement for the position.

Field Descriptions

The fields of the GC_MAKECALL_BLK data structure are described as follows:

gclib

A pointer that points to information used by the **gc_MakeCall( )** function that is common across technologies. The GCLIB_MAKECALL_BLK structure supports generic call related parameters. The following GCLIB_MAKECALL_BLK structure shows the fields that are common across most protocols. In cases where a protocol does not require changing any one of the fields, a default value will be assigned.

```
typedef struct
{
   GCLIB_ADDRESS_BLK destination;  /* Called party information */
   GCLIB_ADDRESS_BLK origination;  /* Calling party information*/
   GCLIB_CHAN_BLKP   chan_info;    /* Pointer to channel information */
   GCLIB_CALL_BLK    call_info;    /* Call information */
   GC_PARM_BLK       ext_datap;    /* Pointer to extended parameters */
} GCLIB_MAKECALL_BLK, *GCLIB_MAKECALL_BLKP;
```

For descriptions of the fields in the GCLIB_MAKECALL_BLK structure, refer to the *Dialogic® Global Call API Library Reference*.

There are certain parameters that are ISDN-specific. These parameters can be defined in the ext_datap field which is of type GC_PARM_BLK. Table 55 lists the parameters that can be included.

Table 55. ISDN Call Setup Parameters

Set ID and Parm ID	Description	Supported Values
For Dialogic® Springware Boards: • Set ID: GCIS_SET_BEARERCHNL • Parm ID: GCIS_PARM_TRANSFERMODE	Bearer Channel information transfer mode	ISDN_ITM_CIRCUIT – circuit switch mode
For Dialogic® DM3 Boards: • Set ID: GCSET_CHAN_CAPABILITY • Parm ID: GCPARM_CAPABILITY	Bearer Channel capability	• GCCAPTYPE_AUDIO • GCCAPTYPE_3KHZ_AUDIO • GCCAPTYPE_7KHZ_AUDIO • GCCAPTYPE_VIDEO
For Dialogic® DM3 Boards: • Set ID: GCSET_CHAN_CAPABILITY • Parm ID: GCPARM_TYPE	Bearer Channel type	• GCCAP_DATA_CCITTV110 • GCCAP_DATA_CCITTV120 • GCCAP_DATA_CCITTX31 • GCCAP_AUDIO_G721ADPCM • GCCAP_AUDIO_g711Alaw64k • GCCAP_AUDIO_g711Alaw56k • GCCAP_AUDIO_g711Ulaw64k • GCCAP_AUDIO_g711Ulaw56k • GCCAP_AUDIO_g722_48k • GCCAP_AUDIO_g722_56k • GCCAP_AUDIO_g722_64k • GCCAP_AUDIO_g7231 • GCCAP_AUDIO_g7231_5_3k • GCCAP_AUDIO_g7231_6_3k
For Dialogic® DM3 Boards: • Set ID: GCSET_CHAN_CAPABILITY • Parm ID: GCPARM_RATE	Bearer Channel rate	• GCCAPRATE_64000 • GCCAPRATE_128000 • GCCAPRATE_384000 • GCCAPRATE_1536000 • GCCAPRATE_1920000
For Dialogic® DM3 Boards: • Set ID: GCSET_SET_BEARERCHNL • Parm ID: GCIS_PARM_TRANSFERRATE **For Dialogic® Springware Boards**: • Set ID: GCIS_SET_BEARERCHNL • Parm ID: GCIS_PARM_TRANSFER_RATE	Bearer Channel information transfer rate	**For DM3 Boards**: • BEAR_RATE_64KBPS • BEAR_RATE_128KBPS • BEAR_RATE_384KBPS • BEAR_RATE_1536KBPS • BEAR_RATE_1920KBPS **For Springware Boards**: • BEARER_RATE_64KBPS
For Dialogic® Springware Boards: • Set ID: ISDN_SET_CALL_INFO • Parm ID: ISDN_INFO_ELEMENTS	User Information element	value_buf field of GC_PARM_DATA contains a pointer to IE_BLK (see the IE_BLK reference page) and contains the information element to be sent to the network.
NOTE: The facility_feature_service and facility_coding_value data elements must be paired to support the specific feature or service requested from the network. You need to know what specific feature or service is being used before entering a value for facility_feature_service.		

Table 55. ISDN Call Setup Parameters (Continued)

Set ID and Parm ID	Description	Supported Values
For Dialogic® Springware Boards: • Set ID: GCIS_SET_FACILITY • Parm ID: GCIS_PARM_FACILITY_FEATURESERVICE	Identifies facility request as a feature or a service (See Note below)	One of the following: • ISDN_FEATURE – request is a facility feature. Features are normally used in the facility message after a call is initiated. Features can also be used in the setup message. • ISDN_SERVICE – requested facility is a service. Services can be used at any time in the NSF IE. Service is often used in the setup message to select a specific network service.
For Dialogic® Springware Boards: • Set ID: GCIS_SET_FACILITY • Parm ID: GCIS_PARM_FACILITY_CODINGVALUE	Facility coding value; identifies the specific feature or service provided (See Note below)	One of the following: • ISDN_CPN_PREF – calling party number preferred • ISDN_SDN – AT&T Software Defined Network • ISDN_BN_PREF – Billing number preferred
For Dialogic® DM3 Boards: • Set ID: CCSET_CALLANALYSIS • Parm IDs: - CCPARM_CA_MODE: Configuring call progress analysis on a per call basis - CCPARM_CA_PAMDSPDVAL: Positive answering machine detection (PAMD) speed value - CCPARM_CA_NOANSR: No Answer; the length of time (in 10 ms units) to wait after the first ringback before deciding that the call is not answered - CCPARM_CA_NOSIG: Continuous No Signal; the maximum amount of silence (in 10 ms units) allowed immediately after cadence detection begins. If exceeded, a no ringback is returned. - CCPARM_CA_PAMDFAILURE: PAMD Fail Time; the maximum time (in 10 ms units) to wait for positive answering machine detection (PAMD) or positive voice detection (PVD) after a cadence break) - CCPARM_CA_PAMD_QTEMP: PAMD Qualification Template; specifies which PAMD template to use - CCPARM_CA_PVD_QTEMP: PVD Qualification Template; specifies which positive voice detection (PVD) template to use	Call Progress Analysis (CPA) parameters	See the "Call Progress Analysis when Using Dialogic® DM3 Boards" section in the *Dialogic® Global Call API Programming Guide* for detailed information.
NOTE: The facility_feature_service and facility_coding_value data elements must be paired to support the specific feature or service requested from the network. You need to know what specific feature or service is being used before entering a value for facility_feature_service.		

cclib
: A pointer that points to information used by the **gc_MakeCall( )** function that is specific to a call control library, in this case ISDN.

Example

The following is a sample GC_MAKECALL_BLK initialization for use with Dialogic® Springware Boards:

```
#include "gclib.h"
#include "gcerr.h"
#include "gcisdn.h"

void makecall(LINEDEV linedev)
{
   CRN crn;
   int cclibid;          /* cclib id for gc_ErrorValue() */
   int gc_error;         /* Global Call error code */
   long cc_error;        /* Call Control Library error code */
   char *msg;            /* points to the error message string */
   int timeout = 30;
   char dnis[] = "6343703";

   GC_PARM_BLKP t_pParmBlk=NULL;
   GC_MAKECALL_BLK gc_makecall;

   gc_util_insert_parm_val(&t_pParmBlk, GCSET_CHAN_CAPABILITY, \
        GCPARM_TYPE, sizeof(unsigned char), GCCAPTYPE_AUDIO);
   gc_util_insert_parm_val(&t_pParmBlk, GCSET_CHAN_CAPABILITY, \
        GCPARM_CAPABILITY, sizeof(unsigned char), 0xFF);
   gc_util_insert_parm_val(&t_pParmBlk, GCSET_CHAN_CAPABILITY, \
        GCPARM_RATE, sizeof(unsigned char), 0xFF);

   gc_util_insert_parm_val(&t_pParmBlk, GCIS_SET_BEARER_CHNL,  \
        GCIS_PARM_TRANSFER_MODE, sizeof(unsigned char), ISDN_ITM_CIRCUIT);
   gc_util_insert_parm_val(&t_pParmBlk, GCIS_SET_BEARER_CHNL, \
        GCIS_PARM_TRANSFER_RATE, sizeof(unsigned char), BEAR_RATE_64KBPS);

   gc_util_insert_parm_val(&t_pParmBlk, GCIS_SET_FACILITY, \
        GCIS_PARM_FACILITY_FEATURESERVICE, sizeof(unsigned char),
             ISDN_SERVICE);
   gc_util_insert_parm_val(&t_pParmBlk, GCIS_SET_FACILITY, \
        GCIS_PARM_FACILITY_CODINGVALUE, sizeof(unsigned char), ISDN_MEGACOM);

   if ((gc_makecall.gclib =
         (GCLIB_MAKECALL_BLKP)malloc(sizeof(GCLIB_MAKECALL_BLK)+
                        t_pParmBlk->parm_data_size))==NULL)
   {
       /* print_error("could not malloc GCLIB_MAKECALL_BLK!\n"); */
       exit(1);
   }

   gc_makecall.gclib->ext_data.parm_data_size = t_pParmBlk->parm_data_size;
   memcpy(gc_makecall.gclib->ext_data.parm_data_buf, t_pParmBlk->parm_data_buf, \
                        t_pParmBlk->parm_data_size);
   gc_makecall.cclib = NULL;
   gc_util_delete_parm_blk(t_pParmBlk);

   gc_makecall.gclib->destination.address_type       = GCADDRTYPE_NAT;
   gc_makecall.gclib->destination.address_plan       = GCADDRPLAN_ISDN;
   gc_makecall.gclib->destination.sub_address_type   = GCSUBADDR_USER;
   gc_makecall.gclib->destination.sub_address_plan = 0;
   strcpy(gc_makecall.gclib->destination.sub_address, "456");
```

```
    gc_makecall.gclib->origination.address_type       = GCADDRTYPE_NAT;
    gc_makecall.gclib->origination.address_plan       = GCADDRPLAN_ISDN;
    gc_makecall.gclib->origination.sub_address_type   = GCSUBADDR_USER;
    gc_makecall.gclib->origination.sub_address_plan   = 0;
    strcpy(gc_makecall.gclib->origination.address, "6346666");
    strcpy(gc_makecall.gclib->origination.sub_address, "456");

    gc_makecall.gclib->call_info.address_info = GCADDRINFO_ENBLOC;

    if(gc_MakeCall(linedev, &crn, dnis, &gc_makecall, timeout, \
              EV_ASYNC) !=  GC_SUCCESS) {
         /* process error return as shown */
         gc_ErrorValue( &gc_error, &cclibid, &cc_error);
         gc_ResultMsg( LIBID_GC, (long) gc_error, &msg);
    }
}
```

IE_BLK

```
typedef struct {
   short   length;                 /* must be less than MAXLEN_IEDATA  */
   char    data[MAXLEN_IEDATA];    /* application defined data */
} IE_BLK, *IE_BLK_PTR;
```

■ Description

The IE_BLK data structure is used to set up and send and receive information to and from the B channel using the **gc_SetInfoElem( )** or the **gc_SndMsg( )** function. The cclib field of the GC_IE_BLK structure (defined in the *Dialogic® Global Call API Library Reference*) uses the IE_BLK structure to define the Information Element (IE) block to be sent using the **gc_SetInfoElem( )** or **gc_SndMsg( )** function.

■ Field Descriptions

The fields of the IE_BLK data structure are described as follows:

length
: Length of data block in bytes. The value must be less than MAXLEN_IEDATA as defined in the *gcisdn.h* header file.

data[MAXLEN_IEDATA]
: Data for user's IE block. Must be formatted to meet CCITT recommendations. The maximum length of the data field is MAXLEN_IEDATA.

L2_BLK

```
typedef struct
{
   char sapi;
   char ces;
   short length;
   char data[MAXLEN_DATA];
} L2_BLK, *L2_BLK_PTR;
```

Description

The L2_BLK data structure is used to send or receive a frame of information to or from the data link layer using the **gc_SndFrame( )** or **gc_GetFrame( )** function. See example code for these functions in the *Dialogic® Global Call API Library Reference* for details.

Field Descriptions

The fields of the L2_BLK data structure are described as follows:

sapi
: Service access point ID (always set to 0).

ces
: Connection endpoint suffix.

> ***Note:*** When using Dialogic® DM3 Boards, the ces field must be set to 1 before the **gc_GetFrame( )** and **gc_SndFrame( )** functions can be used to get and send Layer 2 frames respectively. When using Dialogic® Springware Boards, the ces field must always be set to 0.

length
: Length of data block in bytes. The value must be less than MAXLEN_IEDATA as defined in the *gcisdn.h* header file.

data[MAXLEN_IEDATA]
: Data for frame. Must be formatted to meet CCITT recommendations. The maximum length of the data field is MAXLEN_IEDATA.

Example

The following L2 block structure can be passed to the function via the L2_BLK structure.

```
l2_blk_ptr[0] = 0x08;     /* Protocol discriminator */
l2_blk_ptr[1] = 0x02;     /* CRN length - 2 bytes */
l2_blk_ptr[2] = 0x03;     /* CRN = 8003 */
l2_blk_ptr[3] = 0x80;
l2_blk_ptr[4] = 0x6e;     /* msg type = NOTIFY */

/* The first IE */
l2_blk_ptr[5] = 0x27;     /* IE type = 27 (NOTIFY) */
l2_blk_ptr[6] = 0x01;     /* The length of NOTIFY */
l2_blk_ptr[7] = 0xF1;     /* Notify indication */
```

```
/* The second IE */
l2_blk_ptr[8] = 0x76;      /* IE type = 76 (REDIRECTION) */
l2_blk_ptr[9] = 0x03;      /* length of redirection */
l2_blk_ptr[10] = 0x01;     /* unknown type and E164 plan */
l2_blk_ptr[11] = 0x03;     /* network provides presentation */
l2_blk_ptr[12] = 0x8D;     /* reason = transfer */
```

NONCRN_BLK

```
typedef struct
{
    char    sapi;
    char    ces;
    short   length;
    char  data[MAXLEN_IEDATA];
} NONCRN_BLK, *NONCRN_BLK_PTR;
```

■ Description

Note: The NONCRN_BLK structure is **not** supported when using Dialogic® DM3 Boards.

The NONCRN_BLK structure contains information related to a GLOBAL or NULL call reference number (CRN).

■ Field Descriptions

The fields of the NONCRN_BLK data structure are described as follows:

sapi
: The Service Access Point Identifier (SAPI). For call control procedures, this value is always zero.

ces
: Connection Endpoint Suffix (CES). For call control procedures, this value is always zero.

length
: The total bytes in the data field.

data
: This field contains the information elements (IEs) to be sent.

SPID_BLK

```
typedef struct
{
   DLINK data_link;
   byte  initializing_term;
   byte  SPID[MAX_SPID_SIZE];
} SPID_BLK;
```

■ Description

Note: The SPID_BLK data structure is **not** supported when using Dialogic® DM3 Boards.

The SPID_BLK data structure is used to cast terminal initialization event data after a CCEV_TERM_REGISTER event is received. The SPID_BLK data structure contains the value of the Service Profile Interface ID (SPID) that is used to determine whether the value is valid for a designated service.

■ Field Descriptions

The fields of the SPID_BLK data structure are described as follows:

data_link
: Data link information. See the DLINK data structure.

initializing_term
: The type of initializing terminal.

SPID
: The Service Profile Interface ID (SPID).

TERM_BLK

```
typedef struct
{
   DLINK data_link;
   byte  ack_type;
   union
   {
      byte cause_value; /* Cause Value if ack type is ISDN_ERROR */
      struct
      {
         byte usid;
         byte tid;
         byte interpreter;
      } uspid;
   } ack_info;
} TERM_BLK, *TERM_BLK_PTR;
```

■ Description

Note: The TERM_BLK data structure is **not** supported when using Dialogic® DM3 Boards.

The TERM_BLK data structure contains information regarding a response to an application request. The response information is passed in a GCEV_SERVICERESP event.

■ Field Descriptions

The fields of the TERM_BLK data structure are described as follows:

data_link
: Data link information. See the DLINK data structure.

ack_type
: The type of acknowledgement to be passed to the firmware. The settings are:
  - ISDN_OK – for a positive acknowledgment
  - ISDN_ERROR – for a negative acknowledgment

cause_value
: The cause value, that is, one of the values defined in the *isdncmd.h* header file. For a list of possible cause values, see Chapter 11, "ISDN-Specific Event Cause Values".

usid
: A User Service Identifier (USID) in the range is 01 to FF. A value of 00 indicates the default.

tid
: A Terminal Identifier (TID) in the range is 01 to 63. A value of 00 specifies that the firmware will determine the value.

interpreter
: Specifies how the usid and tid values are to be interpreted. Possible value settings are:
  - 0 indicates that the terminal is selected when it matches both the USID and TID
  - 1 indicates that the terminal is selected when it matches the USID, but not the TID

TERM_NACK_BLK

```
typedef struct
{
   DLINK data_link;
   byte  cause_value;
} TERM_NACK_BLK;
```

■ Description

Note: The TERM_NACK_BLK data structure is **not** supported when using Dialogic® DM3 Boards.

The TERM_NACK_BLK data structure is used to cast terminal initialization event data after a CCEV_RCVTERMREG_NACK event is received. The TERM_NACK_BLK data structure contains the cause value for the event, indicating why the terminal initialization request was rejected by the network.

■ Field Descriptions

The fields of the TERM_NACK_BLK data structure are described as follows:

data_link
: Data link information. See the DLINK data structure.

cause_value
: A value that indicates why the terminal initialization request was rejected by the network. Table 56 lists the possible cause values that may be returned in the TERM_NACK_BLK data structure after receiving a CCEV_RCVTERMREG_NACK event. Any values provided by the Network that are not listed in the table are also be passed to the application.

Table 56. Cause Values Associated with CCEV_RCVTERMREG_NACK

Cause Value	Q.850 Description	Meaning
0x26	Network out of order	Used when the network has removed the TEI, causing the data link to go down.
0x63	Information element/parameter non-existent or not implemented	Switch does not support endpoint initialization.
0x64	Invalid information element contents	SPID was most likely coded incorrectly.
0x66	Recovery on timer expiry	Application tried two attempts at initialization with no response from the network.
0x6F	Protocol error, unspecified	Used when no cause was given for the rejection.

ToneParm

```
Struct toneParm
{
    uint16    duration;    //1 ~ 65535 (in 10 ms, 0xffff - forever)
    uint16    freq1;       //200 ~ 3100 Hz
    int16     amp1;        //-40 ~ +3 dB
    uint16    freq2;       //200 ~ 3100 Hz
    int16     amp2;        //-40 ~ +3 dB
    uint16    toneOn1;     //1 ~ 65535 (in 10 ms)
    uint16    toneOff1;    //0 ~ 65534 (in 10 ms)
    uint16    reserv1;     //reserved for future use
    uint16    reserv2;     //reserved for future use
}
```

■ Description

Note: The ToneParm data structure is **not** supported when using Dialogic® DM3 Boards.

The ToneParm data structure is used to redefine a firmware-applied tone's attributes using the **cc_ToneRedefine( )** function or to play a user-defined tone using the **cc_PlayTone( )** function.

Note: The Dialogic® Global Call API does not provide functions for tone management. The ISDN call control library functions **cc_ToneRedefine( )**, **cc_PlayTone( )**, and **cc_StopTone( )** are appropriate in this context. However, the use of the ISDN call control library is not officially supported and the *Dialogic® ISDN Software Reference*, in which these functions are documented, may not be included in the documentation for future Dialogic® Software Releases.

■ Field Descriptions

The fields of the ToneParm data structure are described as follows:

duration
: Specifies the duration of the tone in 10 ms units. The range is 1 to 65535. Set to -1 to play forever.

freq1
: Specifies the frequency of the tone. The range is 200 to 3100 Hz.

amp1
: Specifies the amplitude of the tone. The range is -40 to +3 dB.

freq2
: Specifies the frequency of the tone. The range is 200 to 3100 Hz.

amp2
: Specifies the amplitude of the tone. The range is -40 to +3 dB.

toneOn1
: Specifies the tone interval, in 10 ms units. The range is 1 to 65535 ms. Set to 1 or greater for continuous tone.

toneOff1
: Specifies the tone interval, in 10 ms units. The range is 0 to 65534 ms. Set to 0 to play a continuous tone.

reserv1
: Reserved for future use.

reserv2
: Reserved for future use.

USPID_BLK

```
typedef struct
{
   DLINK data_link;
   struct
   {
      byte usid;
      byte tid;
      byte interpreter;
   } uspid;
} USPID_BLK;
```

■ Description

Note: The USPID_BLK data structure is **not** supported when using Dialogic® DM3 Boards.

The USPID_BLK data structure is used to cast terminal initialization event data after a CCEV_RCVTERMREG_ACK event is received. The USPID_BLK data structure contains the value of a valid User Service Profile Interface.

■ Field Descriptions

The fields of the USPID_BLK data structure are described as follows:

data_link
: Data link information. See the DLINK data structure for more information.

uspid.usid
: A User Service Identifier (USID) in the range 01 to FF. A value of 00 indicates the default.

uspid.tid
: A Terminal Identifier (TID) in the range 01 to 63. A value of 00 specifies that the firmware will determine the value.

uspid.interpreter
: Specifies how the usid and tid values are to be interpreted. Possible value settings are:

- 0 indicates that the terminal is selected when it matches both the USID and TID
- 1 indicates that the terminal is selected when it matches the USID, but not the TID

USRINFO_ELEM

```
typedef struct {
   unsigned char length;    /* protocol_discriminator + user information length */
   unsigned char protocol_discriminator;
   char usrinformation[256];
} USRINFO_ELEM, *USRINFO_ELEM_PTR;
```

■ Description

The USRINFO_ELEM data structure is used to return User-to-User Information (UUI) data when using the **gc_GetCallInfo( )** or **gc_GetSigInfo( )** function.

■ Field Descriptions

The fields of the USRINFO_ELEM data structure are described as follows:

length
: Defines the length of the data block in bytes. Value must be the sum of the protocol_discriminator length plus the usrinformation length.

protocol_discriminator
: Defines the network protocol.

usrinformation
: Data containing the application dependent user information.

ISDN-Specific Event Cause Values — 11

This chapter lists the supported Integrated Services Digital Network (ISDN)-specific event cause values and provides a description of each value. The cause values are different for Dialogic® DM3 Boards and Dialogic® Springware Boards, and are categorized as follows based on the origin of the cause value:

- Network cause values
- Call control library cause values
- Firmware-related cause values

Network Cause Values When Using Dialogic® DM3 Boards

Table 57 shows the valid network cause values for the various supported protocols when using Dialogic® DM3 Boards.

Table 57. Network Cause Values When Using Dialogic® DM3 Boards

Cause Value (Decimal)	Cause Value (Hex)	Description and Define	4ESS	5ESS	DMS100	NI2	NET5	NTT
01	0x01	Unassigned (unallocated) number UNASSIGNED_NUMBER	✓	✓	✓	✓	✓	✓
02	0x02	No route to specified transit network NO_ROUTE	✓	✓	✓	✓	✓	✓
03	0x03	No route to destination NO_ROUTE_TO_DEST		✓		✓	✓	✓
06	0x06	Channel unacceptable CHANNEL_UNACCEPTABLE	✓				✓	✓
07	0x07	Call awarded in established channel CALL_AWARDED_IN_EST_CHAN					✓	✓
16	0x10	Normal call clearing NORMAL_CLEARING	✓	✓	✓	✓	✓	✓
17	0x11	User busy USER_BUSY	✓	✓	✓	✓	✓	✓
18	0x12	No user responding NO_USER_RESPONDING	✓	✓	✓	✓	✓	✓
19	0x13	No answer from user NO_ANSWER_FROM_USER		✓		✓	✓	✓

Note: The cause values in this table are ORed with the value ERR_ISDN_CAUSE (0x200) which identifies them as network cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.

Table 57. Network Cause Values When Using Dialogic® DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description and Define	4ESS	5ESS	DMS100	NI2	NET5	NTT
21	0x15	Call rejected CALL_REJECTED	✓	✓	✓	✓	✓	✓
22	0x16	Number changed NUMBER_CHANGED	✓	✓	✓	✓	✓	✓
26	0x1A	Network out of order NON_SELECTED_USR_CLEAR					✓	✓
27	0x1B	Destination out of order DEST_OUT_OF_ORDER		✓		✓	✓	✓
28	0x1C	Invalid number format (incomplete number) INVALID_NUMBER_FORMAT	✓	✓	✓	✓	✓	✓
29	0x1D	Facility rejected FACILITY_REJECTED	✓		✓		✓	✓
30	0x1E	Response to STATUS ENQUIRY RESP_TO_STAT_ENQ	✓	✓	✓	✓	✓	✓
31	0x1F	Normal, unspecified UNSPECIFIED_CAUSE	✓	✓	✓	✓	✓	✓
34	0x22	No circuit/channel available NO_CIRCUIT_AVAILABLE	✓	✓	✓	✓	✓	✓
38	0x26	Network out of order NETWORK_OUT_OF_ORDER	✓				✓	✓
41	0x29	Temporary failure TEMPORARY_FAILURE	✓	✓		✓	✓	✓
42	0x2A	Switching equipment congestion NETWORK_CONGESTION	✓	✓	✓	✓	✓	✓
43	0x2B	Access information discarded ACCESS_INFO_DISCARDED	✓	✓	✓	✓	✓	✓
44	0x2C	Requested circuit/channel not available REQ_CHANNEL_NOT_AVAIL	✓		✓		✓	✓
45	0x2D	Call preempted PRE_EMPTED	✓					
47	0x2F	Resource unavailable RESOURCE_UNAVAILABLE		✓	✓	✓	✓	✓
49	0x31	QoS unavailable QOS_UNAVAILABLE					✓	✓
50	0x32	Requested facility not subscribed (see Q.850) FACILITY_NOT_SUBSCRIBED	✓	✓	✓	✓	✓	✓
52	0x34	Outgoing call barred OUTGOING_CALL_BARRED	✓	✓		✓		

Note: The cause values in this table are ORed with the value ERR_ISDN_CAUSE (0x200) which identifies them as network cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.

Table 57. Network Cause Values When Using Dialogic® DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description and Define	4ESS	5ESS	DMS100	NI2	NET5	NTT
54	0x36	Incoming call barred INCOMING_CALL_BARRED	✓	✓	✓	✓		
57	0x39	Bearer capability not authorized BEAR_CAP_NOT_AUTHL					✓	✓
58	0x3A	Bearer capability not presently available BEAR_CAP_NOT_AVAIL	✓	✓	✓	✓	✓	✓
63	0x3F	Service or option not available, unspecified SERVICE_NOT_AVAIL	✓		✓		✓	✓
65	0x41	Bearer capability not implemented CAP_NOT_IMPLEMENTED	✓	✓	✓	✓	✓	✓
66	0x42	Channel type not implemented CHAN_NOT_IMPLEMENTED	✓	✓	✓	✓	✓	✓
69	0x45	Requested facility not implemented FACILITY_NOT_IMPLEMENT	✓	✓		✓	✓	✓
70	0x46	Restricted digit information only RESTRICTED_DIG_INFO_ONLY			✓		✓	✓
79	0x4F	Service not implemented SERVICE_NOT_IMPLEMENTED			✓		✓	✓
81	0x51	Invalid call reference value INVALID_CALL_REF	✓	✓	✓	✓	✓	✓
82	0x52	Identified channel does not exist CHAN_DOES_NOT_EXIST	✓	✓	✓	✓	✓	✓
83	0x53	Bad call ID for suspended call BAD_CALL_ID_FOR_SUS_CALL					✓	✓
84	0x54	Call ID not in use CALL_ID_NOT_IN_USE					✓	✓
85	0x55	No suspended call NO_SUSPENDED_CALL					✓	✓
86	0x56	Call ID cleared CALL_ID_CLEARED					✓	✓
88	0x58	Incompatible destination INCOMPATIBLE_DEST	✓	✓	✓	✓	✓	✓
90	0x5A	Nonexistent CUG NONEXISTENT_CUG			✓			
91	0x5B	Invalid transmission network INVALID_TRANS_NETWORK					✓	✓
95	0x5F	Invalid message, unspecified INVALID_MSG_UNSPEC			✓		✓	✓

Note: The cause values in this table are ORed with the value ERR_ISDN_CAUSE (0x200) which identifies them as network cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.

Table 57. Network Cause Values When Using Dialogic® DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description and Define	4ESS	5ESS	DMS100	NI2	NET5	NTT
96	0x60	Mandatory information element is missing MANDATORY_IE_MISSING	✓	✓	✓	✓	✓	✓
97	0x61	Message type non-existent or not implemented NONEXISTENT_MSG	✓	✓	✓	✓	✓	✓
98	0x62	Message not compatible with call state or message type non-existent or not implemented WRONG_MESSAGE	✓	✓		✓	✓	✓
99	0x63	Information element non-existent or not implemented BAD_INFO_ELEM			✓		✓	✓
100	0x64	Invalid information element contents INVALID_ELEM_CONTENTS	✓	✓	✓	✓	✓	✓
101	0x65	Message not compatible with call state WRONG_MSG_FOR_STATE			✓		✓	✓
102	0x66	Recovery on time expiry TIMER_EXPIRY	✓	✓	✓	✓	✓	✓
103	0x67	Invalid length for information element MANDATORY_IE_LEN_ERR					✓	
111	0x67	Protocol error, unspecified PROTOCOL_ERROR			✓		✓	✓
127	0x7F	Interworking, unspecified INTERWORKING_UNSPEC	✓	✓	✓	✓	✓	✓

Note: The cause values in this table are ORed with the value ERR_ISDN_CAUSE (0x200) which identifies them as network cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.

Call Control Library Cause Values When Using Dialogic® DM3 Boards

Table 58 lists the ISDN call control library cause values supported by Dialogic® DM3 Boards.

Table 58. Call Control Library Cause Values When Using Dialogic® DM3 Boards

Cause Value (Decimal)	Cause Value (Hex)	Description
128	0x80	Requested information available. No more expected.
129	0x81	Requested information available. More expected.
130	0x82	Some of the requested information available. Timeout.
131	0x83	Some of the requested information available. No more expected.
132	0x84	Requested information not available. Timeout.
133	0x85	Requested information not available. No more expected.
134	0x86	Information has been sent successfully.
Note: The cause values in this table are ORed with the value ERR_ISDN_LIB (0x300) which identifies them as call control library cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.		

Note: In addition to the list above, network cause values from Table 57, "Network Cause Values When Using Dialogic® DM3 Boards", on page 289 can also be sent to the application as call control library causes.

Firmware-Related Cause Values When Using Dialogic® DM3 Boards

The following cause value is supported for the category identified by ERR_ISDN_FW (0x100) (see Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162):

WRONG_MSG_FOR_STATE (0x65)
Cause 101: Message not compatible with call state

In addition, the cause code values in Table 59 are supported for the category identified by NON_ISDN_CAUSE (0x0c0) (see Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162).

Table 59. Firmware-Related Cause Values When Using Dialogic® DM3 Boards

Cause Value (Decimal)	Cause Value (Hex)	Description
01	0x01	Busy
02	0x02	Call Completion
03	0x03	Canceled
Note: The cause values in this table are ORed with the value NON_ISDN_CAUSE (0xC0) which identifies them as firmware-related cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.		

Table 59. Firmware-Related Cause Values When Using Dialogic® DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description
04	0x04	Network congestion
05	0x05	Destination busy
06	0x06	Bad destination address
07	0x07	Destination out of order
08	0x08	Destination unreachable
09	0x09	Forward
10	0x0A	Incompatible
11	0x0B	Incoming call
12	0x0C	New call
13	0x0D	No answer from user
14	0x0E	Normal clearing
15	0x0F	Network alarm
16	0x10	Pickup
17	0x11	Protocol error
18	0x12	Redirection
19	0x13	Remote termination
20	0x14	Call rejected
21	0x15	Special Information Tone (SIT)
22	0x16	SIT Custom Irregular
23	0x17	SIT No Circuit
24	0x18	SIT Reorder
25	0x19	Transfer
26	0x1A	Unavailable
27	0x1B	Unknown cause
28	0x1C	Unallocated number
29	0x1D	No route
30	0x1E	Number changed
31	0x1F	Destination out of order
32	0x20	Invalid format
33	0x21	Channel unavailable
34	0x22	Channel unacceptable
35	0x23	Channel not implemented
Note: The cause values in this table are ORed with the value NON_ISDN_CAUSE (0xC0) which identifies them as firmware-related cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.		

Table 59. Firmware-Related Cause Values When Using Dialogic® DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description
36	0x24	No channel
37	0x25	No response
38	0x26	Facility not subscribed
39	0x27	Facility not implemented
40	0x28	Service not implemented
41	0x29	Barred inbound
42	0x2A	Barred outbound
43	0x2B	Destination incompatible
44	0x2C	Bearer capability unavailable
45	0x2D	Normal unspecified
46	0x2E	Req timed out
47	0x2F	Remote retrieve
48	0x30	Remote reconnect
49	0x31	Local timeout
50	0x32	Ack
51	0x33	No ringback
52	0x34	SIT vacant circuit
53	0x35	SIT operator intercept
54	0x36	SIT no circuit interlata
55	0x37	SIT reorder interlata
56	0x38	SIT ineffective other
Note: The cause values in this table are ORed with the value NON_ISDN_CAUSE (0xC0) which identifies them as firmware-related cause values. See Section 4.5.1, "ISDN Event Cause Values When Using Dialogic® DM3 Boards", on page 162 for more information.		

Network Cause Values when Using Dialogic® Springware Boards

The following is a list of ISDN network cause values. Each value is followed by a description. The values are listed in alphabetic order.

Note: The cause codes listed below are ORed with the value ERR_ISDN_CAUSE (0x200) which identifies them as network cause values. See Section 4.5.2, "ISDN Event Cause Values When Using Dialogic® Springware Boards", on page 163 for more information.

BAD_INFO_ELEM
Cause 99: Information element non-existent or not implemented

BEAR_CAP_NOT_AVAIL
Cause 58: Bearer capability not presently available

CALL_REJECTED
Cause 21: Call rejected

CAP_NOT_IMPLEMENTED
Cause 65: Bearer capability not implemented

CHAN_DOES_NOT_EXIST
Cause 82: Identified channel does not exist

CHAN_NOT_IMPLEMENTED
Cause 66: Channel type not implemented

CHANNEL_UNACCEPTABLE
Cause 06: Channel unacceptable

DEST_OUT_OF_ORDER
Cause 27: Destination out of order

FACILITY_NOT_IMPLEMENT
Cause 69: Requested facility not implemented

FACILITY_NOT_SUBSCRIBED
Cause 50: Requested facility not subscribed (see Q.850)

FACILITY_REJECTED
Cause 29: Facility rejected

INCOMING_CALL_BARRED
Cause 54: Incoming call barred

INCOMPATIBLE_DEST
Cause 88: Incompatible destination

INTERWORKING_UNSPEC
Cause 127: Interworking, unspecified

INVALID_CALL_REF
Cause 81: Invalid call reference value

INVALID_ELEM_CONTENTS
Cause 100: Invalid information element contents

INVALID_MSG_UNSPEC
Cause 95: Invalid message, unspecified

INVALID_NUMBER_FORMAT
Cause 28: Invalid number format (incomplete number)

MANDATORY_IE_LEN_ERR
Cause 103: Invalid length for information element

MANDATORY_IE_MISSING
Cause 96: Mandatory information element is missing

NETWORK_CONGESTION
Cause 42: Switching equipment congestion

NETWORK_OUT_OF_ORDER
Cause 38: Network out of order

NO_CIRCUIT_AVAILABLE
    Cause 34: No circuit/channel available

NO_ROUTE
    Cause 02: No route to specified transit network

NO_USER_RESPONDING
    Cause 18: No user responding

NONEXISTENT_MSG
    Cause 97: Message type non-existent or not implemented

NORMAL_CLEARING
    Cause 16: Normal call clearing

NUMBER_CHANGED
    Cause 22: Number changed

OUTGOING_CALL_BARRED
    Cause 52: Outgoing call barred

PRE_EMPTED
    Cause 45: Call preempted

PROTOCOL_ERROR
    Cause 111: Protocol error, unspecified

REQ_CHANNEL_NOT_AVAIL
    Cause 44: Requested circuit/channel not available

RESP_TO_STAT_ENQ
    Cause 30: Response to STATUS ENQUIRY

SERVICE_NOT_AVAIL
    Cause 63: Service or option not available, unspecified

TEMPORARY_FAILURE
    Cause 41: Temporary failure

TIMER_EXPIRY
    Cause 102: Recovery on time expiry

UNASSIGNED_NUMBER
    Cause 01: Unassigned (unallocated) number

UNSPECIFIED_CAUSE
    Cause 31: Normal, unspecified

USER_BUSY
    Cause 17: User busy

WRONG_MESSAGE
    Cause 98: Message not compatible with call state or message type non-existent or not implemented

WRONG_MSG_FOR_STATE
    Cause 101: Message not compatible with call state

Call Control Library Cause Values When Using Dialogic® Springware Boards

The following is a list of ISDN call control library cause values. Each value is followed by a description. The values are listed in alphabetic order.

Note: The cause values listed below are ORed with the value ERR_ISDN_LIB (0x300) which identifies them as call control library cause values. See Section 4.5.2, "ISDN Event Cause Values When Using Dialogic® Springware Boards", on page 163 for more information.

E_ABORTED
: Previous task aborted by **gc_ResetLineDev( )** function

E_BADSTATE
: Invalid state

E_FB_UNAVAIL
: Flexible billing unavailable (applies only to the **gc_SetBilling( )** function)

E_ISBADBUFADDR
: Invalid buffer address

E_ISBADCALLID
: Invalid call identifier

E_ISBADCRN
: Invalid call reference number

E_ISBADIF
: Invalid interface number

E_ISBADPAR
: Invalid input parameter(s)

E_ISBADTS
: Invalid time slot

E_ISCONFIG
: Configuration error

E_ISFILEOPENFAIL
: Failed to open a file

E_ISINVNETWORK
: Invalid network type (applies only to the **gc_ReqANI( )** function)

E_ISMAXLEN
: Exceeds maximum length

E_ISNOFACILITYBUF
: Network facility buffer not ready

E_ISNOINFO
: Information not available

E_ISNOINFOBUF
: Information requested by the **gc_GetCallInfo( )** function call is not available

E_ISNOMEM
Out of memory

E_ISNULLPTR
Null pointer error

E_ISREADY
Board not ready

E_ISSUCC
Message acknowledged

E_ISTNACT
Trace is not activated; application either tried to stop a non-existent trace function or to start the trace function twice on the same D channel.

E_TRACEFAIL
Failed to get trace information

E_UNKNOWNRESULT
Unknown result code

Firmware-Related Cause Values When Using Dialogic® Springware Boards

The following is a list of ISDN firmware-related cause values supported by Dialogic® Springware Boards. Each value is followed by a description. The values are listed in alphabetic order.

Note: The cause values listed below are ORed with the value ERR_ISDN_FW (0x100) which identifies them as firmware-related cause values.

ISDN_BADARGU
Invalid internal firmware command argument(s) possibly caused by an invalid function parameter.

ISDN_BADCALLID
Invalid call ID. No call record exists for specified call ID.

ISDN_BADDSL
Wrong DSL (Digital Subscriber Line) number. Will not occur in non-NFAS environment.

ISDN_BADIF
Invalid ISDN interface ID. Will not occur in non-NFAS environment.

ISDN_BADMSG
Unsupported messages for DASS2: ALERTING, CONGESTION, FACILITY, FACILITY_ACKNOWLEDGEMENT, FACILITY_REJECT, UUI, NOTIFY, and RELEASE.

ISDN_BADSERVICE
The requested network service, such as **gc_ReqANI( )** or **gc_SndMsg( )**, is not supported by the network and was rejected.

ISDN_BADSS
Unspecified service state was requested.

ISDN_BADSTATE
Cannot accept the event in the current state.

ISDN_BADSTR
Invalid phone number string. Phone digits string contains an invalid phone digit number.

ISDN_BADTS
Wrong time slot. Will occur when a second call is placed on an already active channel.

ISDN_CFGERR
Configuration error.

ISDN_CHRST_ERR
Channel restart error.

ISDN_INVALID_EVENT
Invalid event for the switch.

ISDN_INVALID_SWITCH_TYPE
Switch type not supported.

ISDN_LINKFAIL
Layer 2 data link failed. Firmware cannot send a message due to Layer 2 data link failure.

ISDN_MISSIE
Missing mandatory IE.

ISDN_NOAVAIL
Out-of-memory, cannot accept a new call request.

ISDN_OK
Normal return code.

ISDN_TSBUSY
Time slot already in use.

Supplementary Reference Information 12

This chapter lists references to publications about Integrated Services Digital Network (ISDN) technology and includes other reference information as follows:

12.1 References to More Information about ISDN Technology

The following publications provide more detailed information on ISDN technology:

- William Stallings, *ISDN and Broadband ISDN with Frame Relay and ATM*, 3rd ed., Prentice Hall, 1995.
- Gerald L. Hopkins, *The ISDN Literacy Book*, Addison Wesley, 1995.
- Hermann J. Helgert, *Integrated Services Digital Networks - Architectures/Protocols/Standards*, Addison Wesley, 1991.
- ISDN Tutorial, *http://www.ralphb.net/ISDN/index.html*.

12.2 DPNSS IEs and Message Types

This section lists the information elements (IEs) and ISDN message types in the ISDN software library that support the DPNSS protocol.

Information Elements for gc_GetCallInfo( ) and gc_GetSigInfo( )

The following tables describe the different types of IEs that can be retrieved for DPNSS using the **gc_GetCallInfo( )** and **gc_GetSigInfo( )** functions.

Table 60. Intrusion IE

Field	Description	Field Selection	Definition
1. IE ID	Busy IE ID	BUSY_IE	Busy IE value for the GCEV_PROCEEDING event indicates that the called party is busy

Table 61. Diversion IE

Field	Description	Field Selection	Definition
1. IE ID	Diversion IE ID	DIVERSION_IE	1. A DIVERSION_IE value in a GCEV_OFFERED event provides information about the diverted from party. 2. A DIVERSION_ IE value in a GCEV_PROCEEDING event provides information about the divert to party.
2. Data	Diversion IE Length	2 + length of Diversion Number	Number of data bytes in this IE
3. Data	Diversion Type	DIVERT_IMMEDIATE DIVERT_ON_BUSY DIVERT_NO_REPLY	Diverted immediately Diverted when called party was busy Diverted when called party did not answer
4. Data	Diversion Location	DIVERT_LOCAL DIVERT_REMOTE	Local diversion Remote diversion
5. Data	Diversion Number	ASCII string	Diverted number

Table 62. Diversion Validation IE

Field	Description	Field Selection	Definition
1. IE ID	Diversion Validation IE ID	DIVERSION_VALIDATION_IE	When this IE is part of a GCEV_OFFERED event, it indicates that the diversion number needs to be validated.

Table 63. Transit IE

Field	Description	Field Selection	Definition
1. IE ID	Transit IE ID	TRANSIT_IE	This IE is received with a GCEV_TRANSIT event.
2. Data	Transit IE Length	Length of Transit data	Number of data bytes in this IE
3. Data	Transit Data	data	Transit data that needs to be sent to the other transfer party

Table 64. Text Display IE

Field	Description	Field Selection	Definition
1. IE ID	Text Display IE ID	TEXT_DISPLAY_IE	This IE can be part of a GCEV_OFFERED event.
2. Data	Text Display IE Length	1 + length of Text Display string	Number of data bytes for this IE

Table 64. Text Display IE (Continued)

Field	Description	Field Selection	Definition
3. Data	Text Display Message Type	TEXT_TYPE_NOT_PRESENT TEXT_TYPE_NAME TEXT_TYPE_MESSAGE TEXT_TYPE_REASON	Associated text is of no particular type Associated text is a name Associated text is a message Associated text is a reason
4. Data	Text Display String	ASCII string	Text Display string. The '*' and '#' symbols cannot be used directly; 0x01 and 0x02 values should be substituted respectively.

Table 65. Network Specific Indications (NSI) IE

Field	Description	Field Selection	Definition
1. IE ID	NSI IE ID	NSI_IE	This IE can be part of any event including the GCEV_NSI event.
2. Data	NSI IE Length	2 + Length of Network Specific Indications (NSI) string	Number of data bytes for this IE
3. Data	NSI Message Type	NSI_EEM NSI_LLM	End-to-end message Link-to-link message
4. Data	NSI String Length	Length of Network Specific Indications (NSI) string	Length of next NSI string
5. Data	NSI String	ASCII string	Network Specific Indications string
Note: NSI IE fields 4 and 5 can be repeated multiple times, as needed.			

Table 66. Extension Status IE

Field	Description	Field Selection	Definition
IE ID	Extension Status IE ID	EXTENSION_STATUS_IE	This IE is used in conjunction with the Virtual Call IE to inquire about the current status of an extension.

Table 67. Virtual Call IE

Field	Description	Field Selection	Definition
IE ID	Virtual Call IE ID	VIRTUALCALL_IE	This IE, when part of a GCEV_OFFERED event, indicates a virtual call.

Information Elements for gc_SetUserInfo( )

The following tables describe the information elements that can be set for DPNSS using the **gc_SetUserInfo**( ) function.

Table 68. Intrusion IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	4	Required value
2. IE ID	Intrusion IE ID	INTRUSION_IE	Use with the **gc_MakeCall( )** function to indicate intrusion privilege.
3. Data	Intrusion IE Length	2	Number of data bytes for this IE
4. Data	Intrusion Type	INTRUDE_PRIOR_VALIDATION INTRUDE_NORMAL	Validate intrusion level prior to intrude Intrude (without validation)
5. Data	Intrusion Level	INTRUSION_LEVEL_1 INTRUSION_LEVEL_2 INTRUSION_LEVEL_3	Intrusion protection level 1 Intrusion protection level 2 Intrusion protection level 3

Table 69. Diversion IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	4 + length of Diversion Number	
2. Data	Diversion IE ID	DIVERSION_IE	Use with the **gc_MakeCall( )** function to indicate why the call was diverted and from where the call was diverted.
3. Data	Diversion IE Length	2 + length of Diversion Number	Number of data bytes for this element
4. Data	Diversion Type	DIVERT_IMMEDIATE DIVERT_ON_BUSY DIVERT_NO_REPLY	Diverted immediately Diverted when called party was busy Diverted when called party did not answer
5. Data	Diversion Location	DIVERT_LOCAL DIVERT_REMOTE	Local diversion Remote diversion
6. Data	Diversion Number	ASCII string	Diverted number

Table 70. Diversion Bypass IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	1	Required value
2. Data	Diversion Bypass IE ID	DIVERSION_BYPASS_IE	Use with the **gc_MakeCall( )** function to indicate that diversion is not allowed.

Table 71. Inquiry IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field		
2. Data	Inquiry IE ID	INQUIRY_IE	Use with the **gc_MakeCall( )** function to indicate three-party call.

Table 72. Extension Status IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	1	Required value
2. Data	Extension Status IE ID	EXTENSION_STATUS_IE	Use in conjunction with the Virtual Call IE to inquire about the current status of an extension.

Table 73. Virtual Call IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	1	Required value
2. Data	Virtual Call IE ID	VIRTUALCALL_IE	Use with the **gc_MakeCall( )** function to indicate virtual call.

Table 74. Text Display IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	3 + length of Text Display String	Required value
2. Data	Text Display IE ID	TEXT_DISPLAY_IE	This IE can be part of a GCEV_OFFERED event.
3. Data	Text Display IE Length	1 + length of Text Display string	Number of data bytes for this information element
4. Data	Text Display Message Type	TEXT_TYPE_NOT_PRESENT TEXT_TYPE_NAME TEXT_TYPE_MESSAGE TEXT_TYPE_REASON	Associated text is of no particular type Associated text is a name Associated text is a message Associated text is a reason
5. Data	Text DISPLAY String	ASCII string	Text Display string. The '*' and '#' symbols cannot be used directly; 0x01 and 0x02 values are substituted respectively.

Table 75. Network Specific Indications (NSI) IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	4 + length of NSI String	Required value
2. Data	NSI IE ID	NSI_IE	Identifies the Network Specific Indications IE
3. Data	NSI IE Length	2 + length of NSI string	Number of data bytes for this IE
4. Data	NSI Message Type	NSI_EEM NSI_LLM	End-to-end message Link-to-link message
5. Data	NSI Length String	Length of Network Specific Indications string	Length of next NSI string
6. Data	NSI String	ASCII string	Network Specific Indications string
Note: NSI IE fields 5 and 6 can be repeated multiple times, as needed.			

DPNSS Message Types for gc_SndMsg( )

The following tables describe the ISDN message types that support the DPNSS protocol.

Table 76. SndMsg_Divert

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	4 + length of Diverted Number	Required value
2. Data	Diversion IE ID	DIVERSION_IE	Identifies the Diversion IE
3. Data	Diversion IE Length	2 + length of Diverted Number	Number of data bytes for this IE
4. Data	Diversion Type	DIVERT_IMMEDIATE DIVERT_ON_BUSY DIVERT_NO_REPLY	Diverted immediately Diverted when called party was busy Diverted when called party did not answer
5. Data	Diversion Location	DIVERT_LOCAL DIVERT_REMOTE	Local diversion Remote diversion
6. Data	Diversion Number	ASCII string	Diverted number

Table 77. SndMsg_Intrude

Field	Description	Field Selection	Definition
1. Length	Total number of bytes of the following data field	3	Required value
2. Data	Intrude IE ID	INTRUDE_IE	Identifies the Intrude IE

Table 77. SndMsg_Intrude (Continued)

Field	Description	Field Selection	Definition
3. Data	Intrude IE Length	1	Number of data bytes for this IE
4. Data	Intrude Type	INTRUDE INTRUDE_WITHDRAW	INTRUDE INTRUDE_WITHDRAW

Table 78. SndMsg_NSI

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	4 + length of NSI String	Required value
2. Data	NSI IE ID	NSI_IE	Identifies the NSI IE
3. Data	NSI IE Length	2 + length of Network Specific Indications (NSI) string	2 + length of Network Specific Indications (NSI) string
4. Data	NSI Message Type	NSI_EEM NSI_LLM	End-to-end message Link-to-link message
5. Data	NSI String Length	Length of Network Specific Indications (NSI) string	Length of next NSI string
6. Data	NSI String	ASCII string	Network Specific Indications string
Note: NSI IE fields 5 and 6 can be repeated multiple times as needed.			

Table 79. SndMsg_Transfer

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	3	Required value
2. Data	Transfer IE ID	TRANSFER_IE	Identifies the Transfer IE
3. Data	Transfer IE Length	1	Number of data bytes for this IE
4. Data	Transfer Direction	TRANSFER_ORIG TRANSFER_TERM	Originating end Terminating end

Table 80. SndMsg_Transit

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	2 + Length of Transit Data	Required value
2. Data	Transit IE ID	TRANSIT_IE	Identifies the Transit IE

Table 80. SndMsg_Transit (Continued)

Field	Description	Field Selection	Definition
3. Data	Transit IE Length	Length of Transit Data	Number of data bytes for this information element
4. Data	Transit Data	data	Transit data received from a GCEV_TRANSIT event

12.3 BRI Supplemental Services

The Dialogic® Global Call API functions allow the implementation of the following supplemental services on BRI boards:

- Call Hold/Retrieve
- Call Transfer
- Called/Calling Party Identification
- Message Waiting
- Subaddressing

Call Hold and Retrieve are invoked using the following API functions (see the appropriate function descriptions in the *Dialogic® Global Call API Library Reference* and in this document for more information):

- **gc_HoldAck( )**
- **gc_HoldCall( )**
- **gc_HoldRej( )**
- **gc_RetrieveAck( )**
- **gc_RetrieveCall( )**
- **gc_RetrieveRej( )**

The other supplemental services are invoked by sending information from the board to the PBX using an appropriate API function. This information is sent as the part of the Layer 3 frame called the Information Element (IE) (see Section 1.3.2, "Framing", on page 23 for more information). In order for the PBX to interpret the IEs as supplemental service requests, the IEs must be sent as Facility Messages.

The following functions can be used to *send* Facility Messages:

gc_Extension( ) with an **ext_id** of CC_EXID_SndMsg
: Sends a call state associated message to the PBX.

gc_Extension( ) with an **ext_id** of CC_EXID_SndNonCallMsg
: Sends a non-Call State related message to the PBX. This function does not require a call reference value.

gc_SetUserInfo( )
: Sets an information element (IE) allowing the application to include application-specific ISDN information elements in the next outgoing message.

The following functions are used to *retrieve* Facility Messages:

gc_GetCallInfo( )
Retrieves the information elements associated with the CRN.

gc_Extension( ) with **ext_id** as CC_EXID_GetNonCallMsg
Retrieves a non-Call State related ISDN messages to the PBX.

The **gc_Extension( )** with an **ext_id** of CC_EXID_SndMsg and **CC_EXID_SndNonCallMsg( )** functions can be used to send Facility Messages or Notify Messages to the PBX. The Facility Message (as defined in ETS 300-196-1) is composed of the following elements:

- Protocol discriminator
- Call reference
- Message type
- Facility Information Element

The supplemental service to be invoked and its associated parameters are specified in the Information Element. This information is PBX-specific and should be provided by the PBX manufacturer. Facility Messages are sent using the **gc_Extension( )** function with an **ext_id** of CC_EXID_cc_SndMsg or the **gc_Extension( )** function with an **ext_id** of CC_EXID_SndNonCallMsg and msg_type = SndMsg_Facility. These functions:

1. Format the Facility Message, inserting the protocol discriminator, call reference number (only for **gc_Extension( )** with **ext_id** as CC_EXID_SndMsg), and message type elements
2. Add the Information Element data (stored in an application buffer)
3. Send all the information to the PBX

The PBX, in turn, interprets and acts on the information, and sends a reply to the BRI board.

As an example, to invoke supplemental service 'X', you could use the **gc_Extension( )** function with an **ext_id** of CC_EXID_SndMsg function and msg_type = SndMsg_Facility. The Information Element would be defined in a data structure as follows:

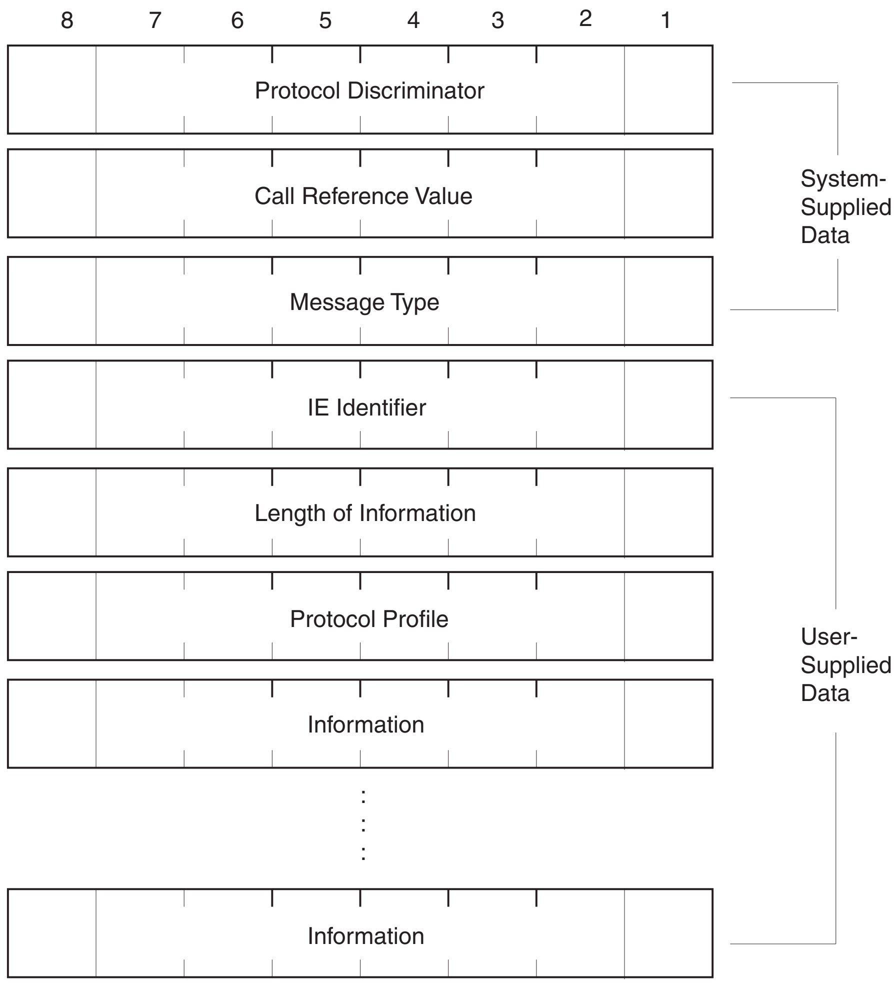
```
ieblk.length = 11;
ieblk.data[0]  = 0x1c;  /* IE Identifier */
ieblk.data[1]  = 0x09;  /* Length of information */
ieblk.data[2]  = 0x91;  /* Protocol Profile */

/* information */
ieblk.data[3]  = 0xa1;  /* Component Type */
ieblk.data[4]  = 0x06;  /* Component Length */
ieblk.data[5]  = 0x02;  /* invoke tag id */
ieblk.data[6]  = 0x01;  /* invode tag length */
ieblk.data[7]  = 0x00;  /* invoke id */
ieblk.data[8]  = 0x02;  /* operation tag */
ieblk.data[9]  = 0x01;  /* operation length */
ieblk.data[10] = 0x06;  /* operation */
```

Note: The information included in the Information Element is dependent on the supplemental service being invoked.

The data sent to the switch would be formatted as follows (Figure 50):

Figure 50. BRI Supplemental Service Information Element Format



Information elements can also be sent using the **gc_SetUserInfo( )** function, which allows the BRI board to send application-specific information elements in the next outgoing message. (For more information, see the **gc_SetUserInfo( )** function description.)

When a supplemental service is invoked, the network may return a NOTIFY message to the user. This message can be retrieved using the **gc_GetCallInfo( )** function.

The Notify message (as defined in ETS 300-196-1) is composed of the following elements:

- Protocol discriminator
- Call reference
- Message type
- Notification Indicator

The Notify message is coded as follows (Figure 51):

Figure 51. BRI Supplemental Services Notify Message Format

8	7	6	5	4	3	2	1
x	x	x	x	x	x	x	x
Protocol Discriminator							
x	x	x	x	x	x	x	x
Call Reference							
x	x	x	x	x	x	x	x
Message Type							
0	0	1	0	0	1	1	1
Notification Indicator Information Element Identifier							
0	0	0	0	1	0	0	1
Length of Notification Indicator Contents							
1/1	x	x	x	x	x	x	x
	Notification Description						
0	x	x	x	x	x	x	x
ext.	Notification Description						
1	0	1	0	0	0	0	1
Notification Data Structure							

Coding requirements for other supported supplemental services are listed in Table 81.

Table 81. ETSI Specification Cross-Reference for Supplemental Services

Supplementary Service/Description	ETS 300 Specification
Explicit Call Transfer - enables a user (user A) to transform two of that user's calls (an active call and a held call), each of which can be an incoming call or an outgoing call, into a new call between user B and user C. The Call Transferred Alerting and Call Transferred Active messages are returned by the network to the user.	367/369/369
Call Hold/ Retrieve - allows a user to interrupt communications on an existing call and then subsequently, if desired, re-establish communications. When on hold, the user may retrieve that call from hold, originate a new call, retrieve another call, or establish connection to an incoming call, for example, a waiting call.	139/140/141
Subaddressing (allows direct connection to individual extensions or devices sharing the same phone number, or, as a proprietary messaging mechanism). Provides additional addressing above the ISDN number of the called user.	059/060/061

Table 81. ETSI Specification Cross-Reference for Supplemental Services (Continued)

Supplementary Service/Description	ETS 300 Specification
Called/Calling Party Identification (CLIP) - Provides the calling user's ISDN number and subaddress information to the called user. This information is sent in the Setup message (see ETS300 102-1) by the calling user to the switch, and from the switch to the called user.	089/091/092
Called/Calling Party Identification (CLIR) - Restricts presentation of the calling user's ISDN number to the called user.	090/091/093
Called/Calling Party Identification (COLP) - Provides the calling user's ISDN number to the called user.	094/096/097
Called/Calling Party Identification (COLR) - restricts the ISDN and the subaddress of the called user.	095/096/098
Advice of Charge - S	178/181/182
Advice of Charge - D	179/181/182
Message Waiting Indication	650/745-1/356-20

Glossary

B channel: A bearer channel that carries the main data.

BRI: Basic Rate Interface. An ISDN service consisting of two 64 kb/s B channels and one 16 kb/s D channel for a total of 144 kb/s.

D channel: A channel that carries control and signaling information.

DPNSS: Digital Private Network Signaling System (DPNSS) is an ISDN protocol. It is not formally regulated, but is a voluntary standard developed by the exchange and large PBX manufacturers, in conjunction with British Telecom, to allow interconnection between their equipment over the ISDN network.

IE: Information Element.

isdiag: An interactive tool used to help verify ISDN line operation and to assist in troubleshooting the network trunk.

ISDNTRACE: A utility that analyzes the binary trace files generated by *isdiag*, an ISDN diagnostics tool.

NCAS: Non-Call Associated Signaling. A facility that allows users to communicate by means of user-to-user signaling without setting up a circuit-switched connection (it does not occupy B channel bandwidth). A temporary signaling connection is established and cleared in a manner similar to the control of a circuit-switch connection.

NFAS: Non-Facility Associated Signaling. The ability to support multiple PRI lines with one 64 kb/s D channel.

NTU: Network Termination Unit. Typically, the first piece of equipment on the customer premises that connects to an ISDN line.

PRI: Primary Rate Interface. An ISDN service consisting of 23 B channels plus one 64 kb/s D channel for a total of 1536 kb/s (T1), or 30 B channels plus one 64 kb/s D channel for a total of 1984 kb/s (E1).

QSIG: The European equivalent of the DPNSS protocol. Unlike DPNSS, it is a regulated standard, but like DPNSS the feature list is optional after the basics of call setup and handling have been implemented. Interoperation of any two switches, therefore, is dependent upon the common features of the implementations of the connected telephone systems.

supplemental services: Services such as call hold and retrieve, call transfer, and message waiting that are considered supplementary to the basic call services provided.

TBCT: Two B Call Transfer. Enables an ISDN PRI user to request the switch to connect together two independent calls on the user's interface.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S