Global Call ISDN Technology User's Guide

for Linux and Windows Operating Systems

05-0653-008

Publication Date: January 2003

Intel Converged Communications
1515 Route 10
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support/

For **Products and Services Information**, visit the Intel Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices** and other contact information, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Table of Contents

List of Figures

List of Tables

How to Use This Guide

This guide is for users who use the Global Call Application Programming Interface (API) to develop Linux or Windows applications in a T-1 or E-1 ISDN (Integrated Services Digital Network) environment. This guide is to be used in conjunction with the *Global Call API Programming Guide* for your operating system and the *Global Call API Library Reference*. Products covered by this guide and the organization of this guide are described in this chapter.

Differences between the implementation of a Global Call application in a Linux or a Windows environment are either described parenthetically or are presented in separate paragraphs or sections.

Products Covered By This Guide

The Integrated Services Digital Network (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

The Global Call software provides a consistent interface across Intel® Dialogic® product interfaces to various ISDN and other networks (for example, T-1 ISDN, E-1 ISDN, E-1 CAS, T-1 robbed bit, analog, and IP). See the release information for your system release software for information on the products that provide ISDN Basic Rate Interface (BRI) and Primary Rate Interface (PRI) capabilities. ISDN PRI products support both T-1 and E-1 ISDN protocols and the transfer of voice and data over T-1 or E-1 trunks. See Chapter 6, “Protocols”, for more on PRI and BRI.

Organization of this Guide

This guide provides information for developing ISDN applications and details specific usage of Global Call functions in ISDN applications as follows:

- Chapter 1, "Developing ISDN Applications", presents guidelines for developing ISDN applications.
- Chapter 2, "Applying Global Call Functions to ISDN Applications", describes the additional functionality of specific Global Call functions used for developing ISDN applications.
- Chapter 3, "Sequence of Function Calls in ISDN", provides asynchronous and synchronous programming models for using Global Call functions in ISDN applications.
- Chapter 4, "ISDN-Specific Parameter Set Reference", provides information about the ISDN-specific parameter sets and parameter IDs used by Global Call.
- Chapter 5, "Data Structure Reference", describes ISDN specific data structures and the elements of these data structures.
- Chapter 6, "Protocols", describes the protocol conventions and programming considerations to be used when incorporating ISDN protocol(s) into your application.
- Chapter 7, "Debugging Applications", describes the diagnostic tools available for debugging ISDN applications in a Global Call environment.
- Appendix A, "Establishing ISDN Connections", contains guidelines for establishing ISDN connections.
- Appendix B, "ISDN Call Scenarios", describes ISDN call scenarios for both asynchronous and synchronous applications.
- Appendix C, "BRI Supplemental Services", describes BRI supplemental services.
- Appendix D, "DPNSS IEs and Message Types", lists the information elements (IEs) and ISDN message types in the ISDN software library that support the DPNSS protocol.
- Appendix E, "DPNSS Call Scenarios", describes common call control scenarios when using the DPNSS protocol.

An index follows the appendices.

1. Developing ISDN Applications

This chapter offers advice and suggestions for programmers designing and coding Global Call applications in a Linux or Windows environment. Specific guidelines for developing ISDN applications are provided. Topics include the following:

- ISDN Features and Benefits
- ISDN Signaling Concepts
- ISDN Connections
- Header Files
- Configuration and Resource Association
- Responding to ISDN Events
- ISDN Extension IDs
- GCEV_EXTENSION Events
- Run Time Configuration Management
- Service Request (BRI Only)
- Alarm Handling
- Error Handling – ISDN Cause Values
- Controlling the Sending of SETUP_ACK and PROCEEDING
- Handling Glare Conditions
- Send and Receive Any IE and Any Message
- Overlap Send
- Direct Layer 2 Access
- D Channel Status
- B Channel Status
- Call Progress and Call Analysis

NOTE: 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.

1.1. ISDN Features and Benefits

The Integrated Services Digital Network (ISDN) is a digital communications network capable of carrying all forms (voice, computer and facsimile) of digitized data 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 Appendix B, "ISDN Call Scenarios", for details):

- **Two B Channel Transfer (TBCT)** – 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 for the 5ESS and 4ESS protocols; see Appendix B, "ISDN Call Scenarios" for details. The feature is also supported by the Q.SIG protocol.
- **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 Appendix B, "ISDN Call Scenarios".
- **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** (AT&T only) – Allows the user to request a caller ID number to identify the origin of the call, when necessary.
- **NFAS** – Non-Facility Associated Signaling that provides support for multiple ISDN spans from a single D channel. See the Release Catalog for your operating system for the products that support the NFAS D channel.

- **DDI (Direct Dialing In)** – DDI service, also called Dialed Number Identification Service (DNIS), 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** – An information element (IE) that may be included 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.

1.2. ISDN Signaling Concepts

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 T-1 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 E-1 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.

Global Call's primary rate implementations 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.

1.2.1. 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 Global Call ISDN products:

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

1.2.2. Data Link Layer (Layer 2) 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 (see Figure 1)

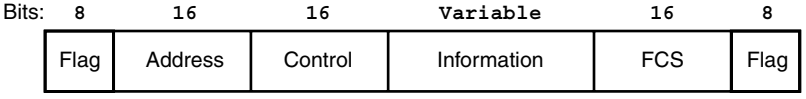


Figure 1. Layer 2 Frame (D Channel)

1.2.3. Network Layer (Layer 3) Frames

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

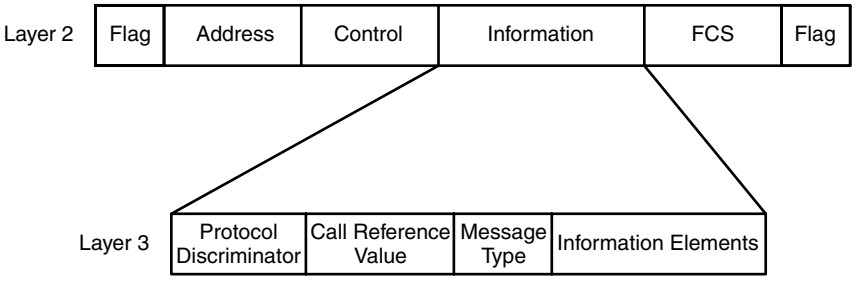


Figure 2. Layer 3 Frame (D Channel)

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 (IE) – used with the message to provide additional information on the type and requirements of the call

1.3. ISDN 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. ISDN vs. 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.”

Table 1. ISDN vs. Analog Connections (Continued)

Step	ISDN Connection	Analog Connection
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 voice library after issuing the **gc_MakeCall( )** function. See also Section 1.5, "Configuration and Resource Association", on page 7.

1.4. Header Files

In addition to the common Global Call header files *gclib.h* and *gcerr.h* that are required irrespective of the technology used, the *gcisdn.h* header file may also be required when developing applications that use ISDN protocols.

NOTE: *gcisdn.h* is only required when the application uses ISDN symbols.

1.5. Configuration and Resource Association

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 Global Call ISDN products, applications can control Primary Rate line connectivity. The 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.

Global Call ISDN products can be placed in a variety of drop-and-insert configurations, providing all the features and benefits of terminate 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 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.

1.6. Responding to ISDN Events

The receipt of an ISDN message or event may require taking the action described in Table 2 to retrieve information or to set up the channel for the next call. The following descriptions supplement the event descriptions listed in the *Global Call API Library Reference* manual.

Table 2. Responding to ISDN Events

Event	Description/Action
GCEV_CALLINFO when using both 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 Springware boards GCEV_CONGESTION event when using 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.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_D_CHAN_STATUS when using both Springware and DM3 boards	Unsolicited ISDN even (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 Springware boards **Note**: Not supported when using 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 and Q.SIG protocols only). • Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_FACILITY when using Springware boards GCEV_FACILITY event when using DM3 boards	Unsolicited ISDN event (not maskable) generated when an incoming Facility Request message is received. • Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_FACILITY_ACK when using Springware boards **Note**: Not supported when using DM3 boards.	Unsolicited ISDN event (not maskable) generated when an incoming FACILITY_ACKNOWLEDGEMENT message is received. • Use **gc_GetCallInfo( )** function to retrieve call information.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_EXTENSION with **ext_id =** GCIS_EXEV_FACILITY_REJ when using Springware boards **Note**: Not supported when using 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 Springware boards **Note**: Not supported when using DM3 boards.	Termination event for ISDN **gc_HoldAck( )** function generated when a Hold Call request is acknowledged successfully. • No action required.
GCEV_HOLDCALL when using Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN event (not maskable) generated when the Hold Call request was acknowledged by the remote end and that the call is in the Hold state (DPNSS and Q.SIG protocols only). • Respond with a **gc_HoldAck( )** or **gc_HoldRej( )** function.
GCEV_HOLDREJ when using Springware boards **Note**: Not supported when using DM3 boards.	Termination event for ISDN **gc_HoldRej( )** function generated when a Hold Call request is rejected successfully. • No action required.
GCEV_EXTENSION with **ext_id =** GCIS_EXEV_L2FRAME when using Springware boards GCEV_L2FRAME event when using DM3 boards	Termination ISDN event (not maskable) generated when an incoming data link layer 2 access message is received. • Use **gc_GetFrame( )** function to retrieve the received frame.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_EXTENSION with **ext_id** = GCIS_EXEV_L2NOBFFR when using Springware boards **Note**: Not supported when using DM3 boards.	Termination 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 Springware boards GCEV_NOTIFY event when using DM3 boards	Termination ISDN 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 Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN 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 Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN event (not maskable) generated when a Network Specific Information (NSI) message is received (DPNSS and Q.SIG protocols only). • Use **gc_GetCallInfo( )** function to retrieve call information.
GCEV_PROCEEDING when using both Springware and DM3 boards	Termination ISDN 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.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_PROGRESSING when using both Springware and DM3 boards	Termination ISDN event (enabled by default) generated when an incoming Progress message is received. • Use **gc_SetEvtMsk( )** function to mask event.
GCEV_REQANI when using Springware boards **Note**: Not supported when using 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 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 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.
GCEV_RETRIEVEACK when using Springware boards **Note**: Not supported when using DM3 boards.	Termination event for ISDN **gc_RetrieveAck( )** function generated when a Retrieve Call request is acknowledged successfully. • No action required.
GCEV_RETRIEVECALL when using Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN event (not maskable), generated when the call is retrieved successfully from the HOLD state (DPNSS and Q.SIG protocols only). • Use the **gc_RetrieveAck( )** or the **gc_RetrieveRej( )** function to respond.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_RETRIEVEREJ when using Springware boards **Note**: Not supported when using DM3 boards.	Termination event for ISDN **gc_RetrieveRej( )** function generated when a Retrieve Call request is rejected successfully (DPNSS and Q.SIG protocols only). • No action required.
GCEV_SETBILLING when using Springware boards **Note**: Not supported when using 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 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 Springware and DM3 boards	Termination ISDN event (enabled by default) generated when an incoming setup ACK (acknowledge) message is received. • No action required.

Table 2. Responding to ISDN Events (Continued)

Event	Description/Action
GCEV_TRANSFERACK when using Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN event (enabled by default) generated when a Transfer acknowledge message is received from the network (DPNSS and Q.SIG protocols only). Indicates that the network accepted a request to transfer a call. • No action required.
GCEV_TRANSFERREJ when using Springware boards **Note**: Not supported when using DM3 boards.	Termination ISDN event (enabled by default) generated when a Transfer Reject message is received from the network (DPNSS and Q.SIG protocols only). Indicates that the network rejected a request to transfer a call. • No action required.
GCEV_TRANSIT when using both Springware and DM3 boards	Termination 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 and Q.SIG protocols only). • No action required.
GCEV_USRINFO when using both Springware and DM3 boards	Termination 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.

1.7. ISDN Extension IDs

Global Call 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 *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 or network event notification is returned via the call control library through an extension event, GCEV_EXTENSION. For more information on the **gc_Extension( )** function and GCEV_EXTENSION, refer to the *Global Call API Programming Guide.*

Table 3 gives 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 3. ISDN Extension IDs

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

Table 3. ISDN Extension IDs

Extension ID	Mode	Termination Event
GCIS_EXID_GETDCHANSTATE when using Springware boards **Note**: When using DM3 boards, D channel state can be retrieved using **gc_GetLineDevState( )**.	Sync	No
GCIS_EXID_GETDLINKSTATE when using Springware boards	Sync,	No
GCIS_EXID_GETENDPOINT when using Springware boards **Note**: When using DM3 boards, retrieving CES and SAPI is not supported.	Sync	No
GCIS_EXID_GETFRAME when using Springware boards **Note**: When using DM3 boards, ISDN frames can be retrieved using **gc_GetFrame( )**.	Sync	No
GCIS_EXID_GETNETCRV when using Springware boards **Note**: When using DM3 boards, the CRV can be retrieved using **gc_GetNetCRV( )**.	Sync	No
GCIS_EXID_GETNONCALLMSG when using Springware boards **Note**: When using DM3 boards, retrieving information associated with the global and null CRN is not supported.	Sync	No
GCIS_EXID_PLAYTONE when using Springware boards **Note**: When using DM3 boards, playing a user defined tone is not supported.	Sync, Async	Yes

Table 3. ISDN Extension IDs

Extension ID	Mode	Termination Event
GCIS_EXID_SETDLINKSTATE when using DM3 and Springware boards **Note**: When using Springware boards, only Sync mode is supported. When using DM3 boards, both the Sync and Async modes are supported; the termination event in Async mode is GCEV_EXTENSION.	Sync, Async	No
GCIS_EXID_SNDFRAME when using Springware boards **Note**: When using DM3 boards, sending frames can be achieved using **gc_SndFrame( )**.	Sync	No
GCIS_EXID_SNDMSG when using Springware boards **Note**: When using DM3 boards, sending a non-call related message can be achieved using **gc_SndMsg( )**	Sync	No
GCIS_EXID_SNDNONCALLMSG when using Springware boards **Note**: When using DM3 boards, sending non-call related messages is not supported.	Sync	No
GCIS_EXID_STOPTONE when using Springware boards **Note**: When using DM3 boards, stopping the playing of a tone is not supported.	Sync, Async	Yes
GCIS_EXID_TONEREDEFINE when using Springware boards **Note**: When using DM3 boards, redefining call progress tone attributes is not supported.	None	Yes

The following sections describe the ISDN extension IDs (**ext_id**) and provide additional information related to their input parameters, as well as cautions, if

applicable, and example codes. The set and parameter IDs are described in Chapter 4, "ISDN-Specific Parameter Set Reference".

1.7.1. Send a Progress Message to the Network

The GCIS_EXID_CALLPROGRESS extension ID is supported when using Springware boards only. When using 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 mode. 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.

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
parmblkp	**set_id** – GCIS_SET_CALLPROGRESS
	parm_id – GCIS_PARM_CALLPROGRESS_INDICATOR
	values –
	• CALL_NOT_END_TO_END_ISDN
	• IN_BAND_INFO
	value_type – int
mode	EV_SYNC

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;
}
```

1.7.2. Retrieve the Status of the B channel

The GCIS_EXID_GETBCHANSTATE extension ID is supported when using Springware boards only. When using DM3 boards, the B channel state can be retrieved using **gc_GetLineDevState( )** function.

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.

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 board
ext_id	GCIS_EXID_GETBCHANSTATE
retblkp	**set_id** – GCIS_SET_CHANSTATE **parm_id** – GCIS_PARM_BCHANSTATE **values** – • ISDN_IN_SERVICE • ISDN_MAINTENANCE • ISDN_OUT_OF_SERVICE **value_type** – int
mode	EV_SYNC

NOTE: This feature is not supported for the BRI/2 board.

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;
}
```

1.7.3. Retrieving the Status of the D channel

The GCIS_EXID_GETBCHANSTATE extension ID is supported when using Springware boards only. When using DM3 boards, the D channel state can be retrieved using **gc_GetLineDevState( )** function.

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

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 board
ext_id	GCIS_EXID_GETDCHANSTATE
retblkp	**set_id** – GCIS_SET_CHANSTATE
	parm_id – GCIS_PARM_DCHANSTATE
	values –
	• DATA_LINK_DOWN
	• DATA_LINK_UP
	value_type – int
mode	EV_SYNC

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

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;
}
```

1.7.4. Retrieve the Logical Data Link State

The GCIS_EXID_GETDLINKSTATE extension ID is supported when using Springware boards only. When using 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.

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** – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – • 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** – • DATA_LINK_DOWN • DATA_LINK_UP • DATA_LINK_DISABLED **value_type** – int
mode	EV_SYNC

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;
}
```

1.7.5. Retrieve the CES and SAPI (BRI Only)

The GCIS_EXID_GETENDPOINT extension ID is supported when using Springware boards only. When using DM3 boards, the GCIS_EXID_GETENDPOINT extension ID is not supported.

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.

The following table provides the parameter inputs for the GCIS_EXID_GETENDPOINT extension ID.

Parameter	Input
target_type	GCTGT_GCLIB_CHAN
target_id	line device handle (linedev) of the device
ext_id	GCIS_EXID_GETENDPOINT
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 – • 0 for BRI • 16 for X.25 packets over D-channel
	value_type – char
mode	EV_SYNC

The GCIS_EXID_GETENDPOINT extension ID applies only to BRI protocols and is not supported for BRI/2 board.

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;
}
```

1.7.6. Retrieve Frame from Application

The GCIS_EXID_GETFRAME extension ID is supported when using Springware boards only. The GCIS_EXID_GETFRAME extension ID is **not** supported when using DM3 boards; use the **gc_GetFrame( )** function instead.

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(**ext_id** = GCIS_EXEV_L2FRAME) event 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).

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** – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type**– char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – • 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.

GCIS_EXID_GETFRAME is not supported for the BRI/2 board or for the PRI DPNSS protocol.

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;
}
```

1.7.7. Retrieve the Network Call Reference Value (CRV)

The GCIS_EXID_GETNETCRV extension ID is supported when using Springware boards only. When using DM3 boards, the CRV can be retrieved using **gc_GetCRV( )** function.

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 Appendix B, "ISDN Call Scenarios" for more information on TBCT.

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

NOTE: This extension ID is not supported for the BRI/2 board

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;
}
```

1.7.8. Retrieve Information for a GLOBAL or NULL CRN Event

The GCIS_EXID_GETNONCALLMSG extension ID is supported when using Springware boards only. When using 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 *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.

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

Parameter	Input
	values – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type** – char
	parm_id – GCIS_PARM_DLINK_SAPI **values** – • 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

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;
}
```

1.7.9. Play a User-Defined Tone

The GCIS_EXID_PLAYTONE extension ID is supported when using Springware boards only. When using DM3 boards, the GCIS_EXID_PLAYTONE extension ID is not supported.

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

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
	parm_id – GCIS_PARM_TONE_FREQ2-
	values – range is 200 to 3100 Hz

Parameter	Input
	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 **ext_id** GCIS_EXEV_PLAYTONE – indicates that the tone was successfully played.
- GCEV_EXTENSION with **ext_id** 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.

This extension ID is not supported for the BRI/2 board or for PRI protocols.

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;
}
```

1.7.10. Set the Logical Data Link State

The GCIS_EXID_SETDLINKSTATE extension ID is supported when using 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 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.

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
parmblkp	the pstruct member of **parmblkp** should point to the DLINK (Data Link Information Block) data structure followed by int. See Section 5.5, "DLINK", on page 189, for a definition of the DLINK structure; see the example code for details. **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES (Springware boards only)

Parameter	Input
	values – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type** – char
	parm_id – GCIS_PARM_DLINK_SAPI (Springware boards only) **values** – • 0 for BRI and PRI • 16 for X.25 packets over D-channel **value_type** – char
	parm_id – GCIS_PARM_DLINK_STATE (DM3 and Springware boards) **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 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

Parameter	Input
mode	EV_ASYNC (DM3 boards only), EV_SYNC (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 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".

Example

NOTE: The following example applies to 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;
}
```

1.7.11. Send Frame to the Data Link Layer

The GCIS_EXID_SNDFRAME extension ID is supported when using Springware boards only. The GCIS_EXID_SNDFRAME extension ID is **not** supported when using DM3 boards; use the **gc_SndFrame( )** function instead.

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).

The following table shows the 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

Parameter	Input
parmblkp	the pstruct member of **parmblkp** should point to the L2_BLK data structure. For a description of the L2_BLK data structure, refer to Section 5.2, "L2_BLK", on page 180. Also see example code for details. **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – • 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.

This extension ID is not supported for the BRI/2 board.

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 */
}
```

1.7.12. Send a Non-Call State Related ISDN Message

The GCIS_EXID_SNDMSG extension ID is supported when using Springware boards only. When using DM3 boards, a non-call state related ISDN message can be sent using the **gc_SndMsg( )** function.

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.

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 messages can be retrieved by calling the **gc_GetCallInfo( )** function. For more information on invoking supplemental services, see Appendix C, "BRI Supplemental Services".

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

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

Parameter	Input
parmblkp	pstruct member of parmblkp should point to the memory block containing integer (msg_type) followed by the IE_BLK data structure. Table 4, below, provides a list of possible values for msg_type. For a description of the IE_BLK data structure, please refer to Section 5.1, "IE_BLK", on page 179. Also see the example code for details. **set_id** – GCIS_SET_SNDMSG **parm_id** – GCIS_PARM_SNDMSGTYPE **values** – All protocols: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus

Parameter	Input
	DPNSS only: • SndMsg_Divert • SndMsg_Intrude • SndMsg_NSI • SndMsg_Transfer • SndMsg_Transit Custom BRI 5ESS only: • 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

Table 4. Possible Values for msg_type

All Protocols	Custom BRI 5ESS only	DPNSS only
SndMsg_Congestion	SndMsg_Drop	SndMsg_Divert
SndMsg_Facility	SndMsg_DropAck	SndMsg_Intrude
SndMsg_FacilityAck	SndMsg_DropRej	SndMsg_NSI
SndMsg_FacilityRej	SndMsg_Redirect	SndMsg_Transfer
SndMsg_Information		SndMsg_Transit
SndMsg_Notify		

Table 4. Possible Values for msg_type (Continued)

All Protocols	Custom BRI 5ESS only	DPNSS only
SndMsg_Status		
SndMsg_StatusEnquiry		
SndMsg_UsrInformation		

Descriptions of the message types for DPNSS are provided in Appendix D, “IEs and ISDN Message Types for DPNSS”.

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;
}
```

1.7.13. Send a Non-Call Related ISDN Message

The GCIS_EXID_SNDNONCALLMSG extension ID is supported when using Springware boards only. When using 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.

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** – • GLOBAL CRN • NULL CRN **value_type** – int **set_id** – GCIS_SET_DLINK **parm_id** – GCIS_PARM_DLINK_CES **values** – • 0 for PRI • 1-8 for BRI when used as a network-side terminal. **value_type** – char **parm_id** – GCIS_PARM_DLINK_SAPI **values** – • 0 for BRI and PRI • 16 for X.25 packets over D-channel **value_type** – char **set_id** – GCIS_SET_SNDMSG

Parameter	Input
	parm_id – GCIS_PARM_SNDMSGTYPE **values** – All protocols: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus DPNSS only: • SndMsg_Divert • SndMsg_Intrude • SndMsg_NSI • SndMsg_Transfer • SndMsg_Transit Custom BRI 5ESS only: • SndMsg_Drop • SndMsg_DropAck • SndMsg_DropRej • SndMsg_Redirect **value_type** – int **set_id** – GCIS_SET_IE **parm_id** – GCIS_PARM_IEDATA **values** – user provided

Parameter	Input
	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.

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;
}
```

1.7.14. Stop Currently Playing Tone (BRI Only)

The GCIS_EXID_STOPTONE extension ID is supported when using Springware boards only. When using DM3 boards, the GCIS_EXID_STOPTONE extension ID is not supported.

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.

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

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;
}
```

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**(GCIS_EXID_STOPTONE) function is issued, the reason for termination will not indicate that gc_Extension(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**(GCIS_EXID_STOPTONE) function from a signal handler, the mode parameter must be set to EV_ASYNC.
5. This function is not supported for the BRI/2 board or PRI protocols.

1.7.15. Redefine Call Progress Tone Attributes (BRI Only)

The GCIS_EXID_TONEREDEFINE extension ID is supported when using Springware boards only. When using DM3 boards, the GCIS_EXID_TONEREDEFINE extension ID is not supported.

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 (see Tone Template Table in *isdnapi.doc*). 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**(GCIS_EXID_TONEREDEFINE) function allows you to redefine the existing tone template, but not the functional meanings of the call progress tones.

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** – • 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

Parameter	Input
	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.

NOTE: This function is not supported for the BRI/2 board or for PRI protocols.

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;
}
```

1.8. GCEV_EXTENSION Events

There are ISDN-specific Global Call 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 1.9, “Run Time Configuration Management”, on page 62.

NOTE: When using DM3 boards, the GCEV_EXTENSION is not supported. 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;
```

The following sections define the different possible extension IDs in the GCEV_EXTENSION event.

Table 5. GCEV_EXTENSION Events

Event	Description
GCIS_EXEV_CONFDROP when using Springware boards **Note**: When using 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 Springware boards When using 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 Springware boards **Note**: When using 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 Springware boards **Note**: When using 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.

Table 5. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_DROPREJ when using Springware boards **Note**: When using 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 Springware boards When using DM3 boards, the equivalent event is GCEV_FACILITY	A FACILITY REQUEST message has been received by the application.
GCIS_EXEV_FACILITY_ACK when using Springware boards **Note**: When using DM3 boards, this event is not supported.	A FACILITY_ACKNOWLEDGEMENT message has been received by the application.
GCIS_EXEV_FACILITY_REJ when using Springware boards **Note**: When using DM3 boards, this event is not supported.	A FACILITY_REJECT message has been received by the application.
GCIS_EXEV_FACILITYGLOBAL when using Springware boards **Note**: When using 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 Springware boards **Note**: When using 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.

Table 5. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_INFOGLOBAL when using Springware boards **Note**: When using 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 Springware boards **Note**: When using 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 Springware boards **Note**: When using DM3 boards, this event is not supported.	Reserved for future use.
GCIS_EXEV_L2FRAME when using Springware boards When using 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 Springware boards **Note**: When using DM3 boards, this event is not supported.	There are no buffers available to save the incoming frame.
GCIS_EXEV_NOFACILITYBUF when using Springware boards **Note**: When using DM3 boards, this event is not supported.	Facility buffer is not ready
GCIS_EXEV_NOTIFY when using Springware boards When using 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 5. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_NOTIFYGLOBAL when using Springware boards **Note**: When using 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 Springware boards **Note**: When using 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 Springware boards **Note**: When using DM3 boards, this event is not supported.	User IE buffer is not ready
GCIS_EXEV_NSI when using Springware boards **Note**: When using 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 Springware boards **Note**: When using DM3 boards, this event is not supported.	User-defined tone successfully played.
GCIS_EXEV_PLAYTONEFAIL when using Springware boards **Note**: When using DM3 boards, this event is not supported.	Request to play user-defined tone failed.

Table 5. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_PROGRESSING when using Springware boards When using 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 Springware boards **Note**: When using DM3 boards, this event is not supported.	A STATUS message has been received from the network.
GCIS_EXEV_STATUS_ENQUIRY when using Springware boards **Note**: When using DM3 boards, this event is not supported.	A STATUS_ENQ message has been received from the network.
GCIS_EXEV_STOPTONE when using Springware boards **Note**: When using DM3 boards, this event is not supported.	The tone operation was terminated.
GCIS_EXEV_STOPTONEFAIL when using Springware boards **Note**: When using DM3 boards, this event is not supported.	The request to terminate the playing of a tone failed.
GCIS_EXEV_TIMER when using Springware boards **Note**: When using DM3 boards, this event is not supported.	An unsolicited event indicating that a timer has expired.
GCIS_EXEV_TONEREDEFINE when using Springware boards **Note**: When using DM3 boards, this event is not supported.	The tone(s) in the firmware tone template table were successfully redefined.
GCIS_EXEV_TONEREDEFINEFAIL when using Springware boards **Note**: When using DM3 boards, this event is not supported.	The request to redefine tone(s) in the firmware tone template table failed.

Table 5. GCEV_EXTENSION Events (Continued)

Event	Description
GCIS_EXEV_TRANSFERACK when using Springware boards **Note**: When using 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 Springware boards **Note**: When using 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.
GCIS_EXEV_TRANSIT when using Springware boards When using 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 Springware boards When using 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 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.

1.9. Run Time Configuration Management

The Global Call Run Time Configuration Management (RTCM) feature allows the dynamic modification of call control-related elements as well as protocol-related parameters.

NOTE: When developing applications that use DM3 boards, RTCM is not supported. However, some RTCM capabilities are achieved by other means as described in the sections following.

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 below provide a list of ISDN parameters that can be retrieved or updated using the RCTM functions. The table lists 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 set and parameter IDs are described in Chapter 4, "ISDN-Specific Parameter Set Reference".

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

NOTE: This functionality is supported when using Springware boards only; not supported when using DM3 boards.

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

Parameter	Input
	values – • 0 - 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** – • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED **value_type** – int
	parm_id – GCIS_PARM_DLINKCFG_PROTOCOL **values** – • 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)

1.9.2. Set Configuration of Digital Subscriber Loop (BRI Only)

NOTE: This functionality is supported when using Springware boards only; not supported when using DM3 boards.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF

Parameter	Input
target_id	board device handle
GC_PARM_BLK	**set_id** – GCIS_SET_DCHANCFG **parm_id** – GCIS_PARM_DCHANCFG_L2ACCESS **values** – • 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** – • 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** – • 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

Parameter	Input
	parm_id – GCIS_PARM_DCHANCFG_FIRMWARE_FEATUREMASKA **values** – • 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** – • AUTO_TEI_TERMINAL = auto TEI assigning Term • FIXED_TEI_TERMINAL = Fixed TEI assigning Term **value_type** – char
	parm_id – GCIS_PARM_DCHANCFG_FIXEDTEIVALUE **values** – range 0 to 63 **value_type** – char
	parm_id – GCIS_PARM_DCHANCFG_AUTOINITFLAG **values** – • AUTO_INIT_TERMINAL • NON_INIT_TERMINAL **value_type** – char

Parameter	Input
	parm_id – GCIS_PARM_DCHANCFG_SPID **value** – ASCII digit string limited to the digits 0-9 and limited in length to MAX_SPID_SIZE **value_type** – string
	parm_id – • 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)

1.9.3. Set/Retrieve Bearer Channel Information Transfer Capability

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

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** – • GCCAPTYPE_AUDIO • GCCAPTYPE_UNDEFINED • 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)

1.9.4. Set/Retrieve Bearer Channel Information Transfer Mode

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

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** – • ISDN_ITM_CIRCUIT • ISDN_ITM_PACKET **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

1.9.5. Set/Retrieve Bearer Channel Information Transfer Rate

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

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)

Parameter	Input
GC_PARM_BLK	**set_id** – GCIS_SET_BEARERCHNL
	parm_id – GCIS_PARM_TRANSFERRATE
	values –
	• 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)

1.9.6. Set/Retrieve Layer 1 Protocol to Use on Bearer Channel

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

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)

Parameter	Input
GC_PARM_BLK	**set_id** – GCSET_CHAN_CAPABILITY
	parm_id – GCPARM_CAPABILITY
	values –
	• GCCAP_DATA_CCITTV110
	• GCCAP_AUDIO_g711Ulaw64k
	• GCCAP_AUDIO_g711Ulaw56k
	• GCCAP_AUDIO_g711Alaw64k
	• GCCAP_AUDIO_g711Alaw56k}
	• GCCAP_AUDIO_G721ADPCM
	• GCCAP_DATA_nonStandard
	• GCCAP_DATA_nonStandard
	• GCCAP_VIDEO_h261
	• GCCAP_DATA_nonStandard
	• 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)

1.9.7. Set/Retrieve Logical Data Link State

NOTE: This functionality is supported for Springware boards only; not supported when using DM3 boards.

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

Parameter	Input
parmblkp	the pstruct member of **parmblkp** should point to the DLINK (Data Link Information Block) data structure followed by int. See Section 5.5, "DLINK", on page 189, for a definition of the DLINK structure; see the example code for details. **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** - • 0 for BRI and PRI • 16 for X.25 packets over D-channel **value_type** - char **parm_id** - GCIS_PARM_DLINKCFG_STATE **values** - • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED **value_type** - int
retblkp	pointer to the buffer containing the requested data link state value. **parm_id** - GCIS_PARM_DLINKCFG_STATE **values** - • DATA_LINK_UP • DATA_LINK_DOWN • DATA_LINK_DISABLED

Parameter	Input
	value_type - int
mode	EV_ASYNC, EV_SYNC

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

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

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** – • GCCAPRATE_EINI460 • GCCAPRATE_600 • GCCAPRATE_1200 • GCCAPRATE_2400 • GCCAPRATE_3600 • GCCAPRATE_4800 • GCCAPRATE_7200 • GCCAPRATE_8000 • GCCAPRATE_9600 • GCCAPRATE_14400

Parameter	Input
	• 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)

1.9.9. Set/Retrieve Called Number Type

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

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** – • 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)

1.9.10. Set/Retrieve Called Number Plan

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

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)

Parameter	Input
GC_PARM_BLK	**set_id** – GCIS_SET_ADDRESS **parm_id** – GCIS_PARM_CALLEDADDRESSPLAN **values** – • 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)

1.9.11. Set/Retrieve Calling Number Type

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

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** – • GCADDRTYPE_INTL • GCADDRTYPE_NAT • GCADDRTYPE_LOC **value_type** – int
mode	EV_SYNC or EV_ASYNC

Parameter	Input
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

1.9.12. Set/Retrieve Calling Number Plan

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

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** – • GCADDRPLAN_UNKNOWN • GCADDRPLAN_ISDN • GCADDRPLAN_TELEPHONY **value_type** – int
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

1.9.13. Set/Retrieve Calling Presentation Indicator

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

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)

1.9.14. Set/Retrieve Calling Screening Indicator

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

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)

1.9.15. Set/Retrieve Multiple IE Buffer Size

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

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

1.9.16. Set SPID number on BRI (North America only)

NOTE: This functionality is supported for Springware boards only; not supported when using DM3 boards.

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 **value** – 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)

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

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

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)

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

NOTE: This functionality is supported for Springware boards only; not supported when using DM3 boards.

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

Parameter	Input
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

1.9.19. Set ISDN-Specific Event Masks

NOTE: This functionality is supported for Springware boards only. When using DM3 boards, ISDN-specific masks can be set using **gc_SetEvtMsk( )**. See Section 2.30, “gc_SetEvtMsk( )”, on page 138 for more information.

Parameter	Input
target_type	GCTGT_CCLIB_CHAN
target_id	line device handle (linedev)

Parameter	Input
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 – **values** – GCISMSK_STATUS † GCISMSK_STATUS_ENQUIRY † GCISMSK_TERMINATE † 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 • GCIS_PARM_SUBMSK **values** and **value_type** – same as GCIS_PARM_ADDMSK above • GCIS_PARM_SETMSK **values** and **value_type** – same as GCIS_PARM_ADDMSK above The following applies only to the **gc_GetConfigData( )** function: • GCIS_PARM_GET_MSK **values** and **value_type** – same as GCIS_PARM_ADDMSK above
mode	EV_SYNC or EV_ASYNC
update condition	GCUPATE_IMMEDIATE (**gc_SetConfigData( )** function only)

1.9.20. 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 Springware boards only. The **gc_SetConfigData( )** function is not supported when using 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_TERMINATE);
     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;
}
```

1.10. Service Request (BRI Only)

NOTE: The following information applies when using Springware boards only. DM3 boards do not support the service request feature.

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 4, "ISDN-Specific Parameter Set Reference".

1.10.1. gc_RespService( )

NOTE: This **gc_RespService( )** function is supported for Springware boards only; not supported when using DM3 boards.

Parameter	Input
target_type	GCTGT_CCLIB_NETIF
target_id	board device handle

Parameter	Input
datap	**set_id** – GCSET_SERVREQ **parm_id** – • PARM_SERVICEID **value** – 0 • PARM_REQTYPE **value** – 0 • 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** – • 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_CAUSEVALUE **values** – • NETWORK_OUT_OF_ORDER • BAD_INFO_ELEM • INVALID_ELEM_CONTENTS • TIMER_EXPIRY • PROTOCOL_ERROR **value_type** – unsigned char • GCIS_PARM_SERVREQ_USID **values** – range is 01 – FF. 00 signifies default **value_type** – unsigned char • GCIS_PARM_SERVREQ_TID **values** – range is 01 – FF. 00 signifies default **value_type** – unsigned char • GCIS_PARM_SERVREQ_INTERPRETER– Specifies how the usid and tid values are to be interpreted. Possible value settings are: **values** – • 0 • 1 • **value_type** – unsigned char
mode	EV_SYNC

NOTE: This function applies only to BRI North American terminal protocols used as the network side. This function is not supported for the BRI/2 board.

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;
}
```

1.10.2. Supported Events

The following Global Call events are supported:

- GCEV_SERVICEREQ
- GCEV_SERVICERESP

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** – • 0 for BRI and PRI • 16 for X.25 packets over D-channel **value_type** – char

Parameter	Input
	set_id – GCIS_SET_DCHANCFG **parm_id** – GCIS_PARM_DCHANCFG_AUTOINITFLAG **values** – • AUTO_INIT_TERMINAL • NON_INIT_TERMINAL **value_type** – char **parm_id** – GCIS_PARM_DCHANCFG_SPID **value** – ASCII digit string limited to the digits 0-9 and limited in length to MAX_SPID_SIZE **value_type** – char

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

Parameter	Input
	values –
	• 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_CAUSE
	values –
	• NETWORK_OUT_OF_ORDER
	• BAD_INFO_ELEM
	• INVALID_ELEM_CONTENTS
	• TIMER_EXPIRYPROTOCOL_ERROR
	value_type – unsigned char
	• GCIS_PARM_SERVREQ_USID
	values – range is 01 – FF. 00 signifies default
	value_type – unsigned char
	• GCIS_PARM_SERVREQ_TID
	values – range is 01 – FF. 00 signifies default
	value_type – unsigned char
	• GCIS_PARM_SERVREQ_INTERPRETER– Specifies how the usid and tid values are to be interpreted. Possible value settings are:
	values –
	• 0
	• 1
	value_type – unsigned char

1.11. Alarm Handling

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

- Alarm Handling for DM3 Boards
- Alarm Handling for Springware Boards

1.11.1. Alarm Handling for DM3 Boards

When using DM3 boards, alarms are recognized on a span 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.

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 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 alarms are blocking and non-blocking as described in the *Global Call API Programming Guide*. However, this capability applies on a span basis only. Changing which alarms are blocking and non-blocking for one timeslot results in changing which alarms are blocking and non-blocking for all time slots on the span.
- For ISDN on T-1 technology, the following is a list of the alarms that can be transmitted:
  - YELLOW
  - BLUE
- For ISDN on E-1 technology, the following is a list of the alarms that can be transmitted:
  - Remote alarm - DEA_REMOTE
  - Unframed all 1’s alarm - DEA_UNFRAMED1

  - Signaling all 1's alarm - DEA_SIGNALALL1
  - Distant multi-framed alarm - DEA_DISTANTMF
- Using the **gc_GetAlarmParm( )** and **gc_SetAlarmParm( )** functions to retrieve and set specific alarm parameters, for example alarm triggers, is not supported.

The following list shows the alarms that are supported for ISDN on E-1 for DM3 boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc_SetAlarmConfiguration( )**.

- DTE1_CRC_CFA† - Time slot 16 CRC failure
- DTE1_CRC_CFAOK - Time slot 16 CRC failure recovered
- DTE1_DCHAN_CFA - D channel out of service
- DTE1_DCHAN_CFAOK - D channel out of service recovered
- 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 alarms that are supported for ISDN on T-1 for DM3 boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc_SetAlarmConfiguration( )**.

- DTT1_DCHAN_CFA - D channel out of service
- DTT1_DCHAN_CFAOK - D channel out of service recovered
- 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_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

1.11.2. Alarm Handling for Springware Boards

As described in the *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 *Global Call API Programming Guide* for more information on GCAMS and the *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 following list shows the alarms that are supported for ISDN on E-1 for Springware boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc_SetAlarmConfiguration( )**.

- 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 alarms that are supported for ISDN on T-1 for Springware boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc_SetAlarmConfiguration( )**.

- 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

1.12. Error Handling – ISDN Cause Values

In addition to Global Call cause values that may be retrieved when an error occurs in an ISDN environment, ISDN cause values may also apply. (See Table 6, "Error Location Values", on page 96 to identify the source of the cause and for a cross-reference to tabular listings of various ISDN causes.) These ISDN cause values 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 *Global Call API Programming Guide*.

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 are listed in Table 6.

Table 6. Error Location Values

Error Location	Description
Firmware ERR_ISDN_FW	Indicates a firmware-related cause/error. Firmware errors are listed in the *isdncmd.h* file and in Table 7. **Note**: ISDN Firmware-Related Cause values are supported when using Springware boards only; not supported when using DM3 boards.

Table 6. Error Location Values

Error Location	Description
Network ERR_ISDN_CAUSE	Returned with a GCEV_DISCONNECTED event. Network cause values are listed in the *isdncmd.h* file and in Table 8. The values listed in this table refer to International Telecommunications Union (ITU) Q.931 standards. Not all cause values are universally supported across switch types.
ISDN Library ERR_ISDN_LIB	Indicates an ISDN call control library-related cause/error. ISDN library errors are listed in the *isdnerr.h* file and in Table 9. **Note**: ISDN Call Control Library Cause values are supported when using Springware boards only; not supported when using DM3 boards.

Table 7. ISDN Firmware-Related Cause Values

Cause	Description
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.
Note: ISDN Firmware-Related Cause Values are **not** supported when using DM3 boards.	

Table 7. ISDN Firmware-Related Cause Values (Continued)

Cause	Description
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.
Note: ISDN Firmware-Related Cause Values are **not** supported when using DM3 boards.	

Table 8. ISDN Network Cause Values

Cause	Q.931 Cause Number: Name
UNASSIGNED_NUMBER	Cause 01: Unassigned (unallocated) number
NO_ROUTE	Cause 02: No route to specified transit network
CHANNEL_UNACCEPTABLE	Cause 06: Channel unacceptable
NORMAL_CLEARING	Cause 16: Normal call clearing
USER_BUSY	Cause 17: User busy
NO_USER_RESPONDING	Cause 18: No user responding
CALL_REJECTED	Cause 21: Call rejected
NUMBER_CHANGED	Cause 22: Number changed

Table 8. ISDN Network Cause Values (Continued)

Cause	Q.931 Cause Number: Name
DEST_OUT_OF_ORDER	Cause 27: Destination out of order
INVALID_NUMBER_FORMAT	Cause 28: Invalid number format (incomplete number)
FACILITY_REJECTED	Cause 29: Facility rejected
RESP_TO_STAT_ENQ	Cause 30: Response to STATUS ENQUIRY
UNSPECIFIED_CAUSE	Cause 31: Normal, unspecified
NO_CIRCUIT_AVAILABLE	Cause 34: No circuit/channel available
NETWORK_OUT_OF_ORDER	Cause 38: Network out of order
TEMPORARY_FAILURE	Cause 41: Temporary failure
NETWORK_CONGESTION	Cause 42: Switching equipment congestion
REQ_CHANNEL_NOT_AVAIL	Cause 44: Requested circuit/channel not available
PRE_EMPTED	Cause 45: Call preempted
FACILITY_NOT_SUBSCRIBED	Cause 50: Requested facility not subscribed (see Q.850)
OUTGOING_CALL_BARRED	Cause 52: Outgoing call barred
INCOMING_CALL_BARRED	Cause 54: Incoming call barred
BEAR_CAP_NOT_AVAIL	Cause 58: Bearer capability not presently available
SERVICE_NOT_AVAIL	Cause 63: Service or option not available, unspecified
CAP_NOT_IMPLEMENTED	Cause 65: Bearer capability not implemented
CHAN_NOT_IMPLEMENTED	Cause 66: Channel type not implemented
FACILITY_NOT_IMPLEMENT	Cause 69: Requested facility not implemented
INVALID_CALL_REF	Cause 81: Invalid call reference value
CHAN_DOES_NOT_EXIST	Cause 82: Identified channel does not exist
INCOMPATIBLE_DEST	Cause 88: Incompatible destination
INVALID_MSG_UNSPEC	Cause 95: Invalid message, unspecified
MANDATORY_IE_MISSING	Cause 96: Mandatory information element is missing
NONEXISTENT_MSG	Cause 97: Message type non-existent or not implemented
WRONG_MESSAGE	Cause 98: Message not compatible with call state or message type non-existent or not implemented

Table 8. ISDN Network Cause Values (Continued)

Cause	Q.931 Cause Number: Name
BAD_INFO_ELEM	Cause 99: Information element non-existent or not implemented
INVALID_ELEM_CONTENTS	Cause 100: Invalid information element contents
WRONG_MSG_FOR_STATE	Cause 101: Message not compatible with call state
TIMER_EXPIRY	Cause 102: Recovery on time expiry
MANDATORY_IE_LEN_ERR	Cause 103: Invalid length for information element
PROTOCOL_ERROR	Cause 111: Protocol error, unspecified
INTERWORKING_UNSPEC	Cause 127: Interworking, unspecified

Table 9. ISDN Call Control Library Cause Values

Cause	Description
E_ISSUCC	Message acknowledged
E_ISREADY	Board not ready
E_ISCONFIG	Configuration error
E_ISNOINFO	Information not available
E_ISNOFACILITYBUF	Network facility buffer not ready
E_ISBADBUFADDR	Invalid buffer address
E_ISBADTS	Invalid time slot
E_ISMAXLEN	Exceeds maximum length
E_ISNULLPTR	Null pointer error
E_ISNOMEM	Out of memory
E_ISFILEOPENFAIL	Failed to open a file
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_ISBADPAR	Invalid input parameter(s)
E_ISBADCALLID	Invalid call identifier
Note: ISDN Call Control Library Cause Values are **not** supported when using DM3 boards.	

Table 9. ISDN Call Control Library Cause Values (Continued)

Cause	Description
E_ISBADCRN	Invalid call reference number
E_ISNOINFOBUF	Information requested by the **gc_GetCallInfo( )** function call is not available.
E_ISINVNETWORK	Invalid network type (applies only to the **gc_ReqANI( )** function).
E_FB_UNAVAIL	Flexible billing unavailable (applies only to the **gc_SetBilling( )** function).
E_ISBADIF	Invalid interface number
E_TRACEFAIL	Failed to get trace information
E_UNKNOWNRESULT	Unknown result code
E_BADSTATE	Invalid state
E_ABORTED	Previous task aborted by **gc_ResetLineDev( )** function.
Note: ISDN Call Control Library Cause Values are **not** supported when using DM3 boards.	

1.13. Controlling the Sending of SETUP_ACK and PROCEEDING

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

- When using 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 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 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 bit mask 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 DM3 and Springware boards.

NOTE: When using Springware boards, on outbound calls, the GCMSK_SETUP_ACK bit mask value can be used to enable or disable the sending of the GCEV_SETUP_ACK to the application. When using DM3 boards, GCMSK_SETUP_ACK is **not** supported.

1.14. 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 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 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 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 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 DM3 boards, the application receives a GCEV_DISCONNECTED event.

1.15. Send and Receive Any IE and Any Message

When using Springware and DM3 boards, the Send Any IE (Information Element) and Send Any Message features provided by the **gc_SetInfoElement( )** 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.

1.16. Overlap Send

When using 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 55, "Modifiable Protocol Parameters", on page 211 for more information.

When using 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 DM3 products provided with the system release software 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.

The **gc_SendMoreInfo( )** function is not supported, so to send extra digits, the application should wait for the GCEV_SETUP_ACK event, which indicates the inbound side acknowledges the SETUP message, construct an IE block containing the digits to be sent, and then call **gc_SndMsg(GlobalCallDeviceHandle, CRN, SndMsg_Information, &IEBlock)** to send the digits. Even after sending a number of digits greater than **CalledNumberCount**, the Sending Complete IE is not sent automatically.

The following is an example of how to send extra digits using overlap send:

```
void mdfSendOverlap(CH_INFO_PTR chanInfop, char* digits)
{
    IE_BLK info;
    GC_IE_BLK gcInfo;
    char length;
    unsigned char type = 0x00;
    unsigned char plan = 0x00;
    for(length = 0; ; length++, digits++)
    {
        if(*digits)
        {
            info.data[3 + length]= *digits & 0x7F; // Bit 8 set to 0
        }
        else
        {
            break;
        }
    }
    info.data[2] = 0x80|plan|(type<<4); // Octet 3: Number Type + Numbering Plan
    info.data[1] = length + 1; // Octet 2: Element Length
    info.data[0] = 0x70; // Octet 1: Called Number ID
    info.length = length + 3; // Information block
    gcInfo.gclib = NULL;
    gcInfo.cclib = &info;
    if(gc_SndMsg(chanInfop->hGC, chanInfop->crn, SndMsg_Information, &gcInfo)< 0)
    {
        mdfError(EGCALL, chanInfop, "gc_SndMsg (SndMsg_Information) failed "
                 "CRN: 0x%X", chanInfop->crn);
    }
}
```

NOTE: Any changes to the CONFIG file for a particular protocol requires the regeneration of the FCD file and the subsequent downloading of the firmware to the boards. The FCDGEN tool, available in the *dialogic\bin* directory, is used to convert a CONFIG file to an FCD file.

1.17. Direct Layer 2 Access

When using 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 55, "Modifiable Protocol Parameters", on page 211 for more information.

When using 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: Any changes to the CONFIG file for a particular protocol requires the regeneration of the FCD file and the subsequent downloading of the firmware to the boards. The FCDGEN tool, available in the *dialogic\bin* directory, is used to convert a CONFIG file to an FCD file. For more information, see the configuration information for DM3 products provided with the system release software.

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

1.18. D Channel Status

When using 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 Springware boards, when the initial D channel status was UP, no initial event was generated. When using 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 *Global Call API Library Reference* for more information.

1.19. B Channel Status

When using 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 when a system is initialized. Thereafter, when the application issues **gc_WaitCall( )** the channel (line device) is placed into service. If **gc_ResetLineDev( )** is subsequently issued, the channel is placed out of service until the application issues **gc_WaitCall( )** again.

Also, on channel devices, if **gc_WaitCall( )** is not issued but **gc_MakeCall( )** is issued, the channel is placed into service for the duration of the call. Once the call is released, the channel is once again out of service.

1.20. Call Progress and Call Analysis

When using DM3 boards, Global Call cannot be used for call progress or call analysis. For flexible routing configurations, the **dx_dial( )** method for call analysis must be used, and both pre-connect call progress and post-connect call analysis are available. See the Voice API documentation for more information on call analysis.

2. Applying Global Call Functions to ISDN Applications

This chapter describes the Global Call API functions that have additional functionality or perform differently when used in an ISDN environment. 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 *Global Call API Library Reference*.

NOTE: The ISDN technology-specific information provided in this chapter **must** be used in conjunction with the detailed, generic function descriptions found in the *Global Call API Library Reference*.

In addition to the functions described in this chapter, the following Global Call ISDN functions provide specific support for DPNSS (for both Springware and DM3 boards) and Q.SIG ISDN (for Springware boards) protocol features. The function descriptions for these functions are provided in the *Global Call API Library Reference*.

- **gc_HoldCall( )** - Places active calls on hold.
- **gc_HoldAck( )** - Accepts hold requests from remote equipment.
- **gc_HoldRej( )** - Rejects hold requests from remote equipment.
- **gc_RetrieveAck( )** - Accepts a retrieve-from-hold request from remote equipment.
- **gc_RetrieveCall( )** - Retrieves a call placed on hold from the Hold state.
- **gc_RetrieveRej( )** - Rejects a retrieve-from-hold request from remote equipment.

See Section 3, “Sequence of Function Calls in ISDN”, on page 149, for a description of using the Global Call API functions in synchronous and asynchronous programming models to establish calls in an ISDN environment.

2.1. gc_AcceptCall( )

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. 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 sec.), then this function must be issued to stop the protocol layer 3 timer.

For ISDN, the **rings** parameter is ignored. Instead, the default value is used.

2.2. gc_AnswerCall( )

The **gc_AnswerCall( )** function must be used to complete the call establishment process:

- in asynchronous mode, this function can be called any time after a GCEV_OFFERED, GCEV_ACCEPT or a GCEV_PROGRESSING event is received.
- in synchronous mode, this 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.

When using Springware boards, 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 timeslot using the **gc_ResetLineDev( )** to handle this glare condition.

When using DM3 boards, 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 DM3 boards, the GCEV_DISCONNECTED event is a valid termination event for the **gc_Answer( )** function.

For ISDN, the **rings** parameter is ignored. Instead, the default value is used.

2.3. gc_CallAck( )

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 2.3.1, "Sending First Response to an Incoming Call", on page 111.
- Request additional DNIS (DDI) digits from the network. See Section 2.3.2, "Requesting Additional DNIS Digits", on page 112.

NOTE: B channel negotiation is not currently available.

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;
```

NOTES: **1.** When using Springware boards, by default, the SETUP ACK and CALL PROCEEDING messages are automatically sent by the

firmware. The **gc_SetConfigData( )** function can be use 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 1.9.19, "Set ISDN-Specific Event Masks", on page 81 for more information.

2. When using DM3 boards, by default, the application controls the sending of the SETUP ACK and CALL PROCEEDING messages. The **gc_SetEvtMask( )** function can be use to change the default so that the firmware automatically sends the SETUP ACK and CALL PROCEEDING messages. See Section 2.30, "gc_SetEvtMsk( )", on page 138 for more information.

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 */
```

NOTE: When using DM3 boards, the only info.info_type value supported is DESTINATION_ADDRESS.

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 *Global Call API Library Reference* for more information.

2.4. gc_CallProgress( )

When using Springware boards, the **gc_CallProgress( )** is obsolete. The **gc_Extension( )** function with an **ext_id** of GCIS_EXID_CALLPROGRESS is the recommended equivalent.

When using DM3 boards, the **gc_CallProgress( )** function is not supported. The **gc_SndMsg( )** function is the recommended equivalent.

2.5. gc_DropCall( )

The **gc_DropCall( )** function used in an ISDN environment supports all of the cause values listed in the *Global Call API Library Reference* except for GC_SEND_SIT. In addition, the cause values listed in Table 10 may be used in an ISDN environment.

Table 10. Cause Values, gc_DropCall( ) Function

Cause	Description
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

Table 10. Cause Values, gc_DropCall( ) Function (Continued)

Cause	Description
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

Table 10. Cause Values, gc_DropCall( ) Function (Continued)

Cause	Description
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.

2.6. gc_Extension( )

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 1.7, “ISDN Extension IDs”, on page 15.

2.7. gc_GetANI( )

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 the completion of a **gc_WaitCall( )** function.

2.8. gc_GetBilling( )

When using Springware boards, 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. E-1 based CTR4 service providers provide billing information that cannot be retrieved using **gc_GetBilling( )** function; use the **gc_GetSigInfo( )** function with the **info_id** parameter set to U_IES instead.

When using DM3 boards, the **gc_GetBilling( )** function is not supported. Use the **gc_GetSigInfo( )** function to retrieve billing information.

2.9. gc_GetCallInfo( )

The **gc_GetCallInfo( )** function gets the Information Elements (IE) 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.

NOTE: When using DM3 boards, since the **gc_GetCallInfo( )** function does not queue information, it is recommended to use the **gc_GetSigInfo( )** function.

The IE_BLK data structure should be used to retrieve unprocessed IEs. See Section 5.1, "IE_BLK", on page 179.

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 Section 5.4, “USRINFO_ELEM”, on page 188. 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 Appendix D, “DPNSS IEs and Message Types” and Appendix E, “DPNSS Call Scenarios” for more information.

NOTE: When using DM3 boards, ISDN billing information (AOC) cannot be retrieved using the **gc_GetCallInfo( )** function. Use the **gc_GetSigInfo(U_IES)** function instead. U_IES retrieves all public IEs (as defined by Telenetworks).

2.10. gc_GetConfigData( )

When using Springware boards, the **gc_GetConfigData( )** function supports the Run 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. Please refer to *Global Call API Library Reference* for detail on description of this API and for the ISDN parameters which are retrievable using this API, refer to Section 1.9, “Run Time Configuration Management”, on page 62.

When using DM3 boards, the **gc_GetConfigData( )** is not supported.

2.11. gc_GetDNIS( )

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.

2.12. gc_GetNetCRV( )

When using DM3 boards, the **gc_GetNetCRV( )** function is supported, but must be manually enabled for use. To enable this function:

- For Linux operating systems, add CC.NetCRV Support = 1 in the */usr/dialogic/cfg/cheetah.cfg* file. To subsequently disable this function, remove this line from the CFG file.
- For Windows operating systems, to enable this function, set parameter **NetCRV Support** to 1 in Key HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Cheetah\CC. To subsequently disable this function, remove this line from the CFG file.

2.13. gc_GetParm( )

See Table 14, "Call Setup Parameters When Using gc_SetParm( )", on page 140 for the ISDN parameters that may be retrieved by the **gc_GetParm( )** function.

When using DM3 boards, the following parameters are supported:

- GCPR_MINDIGITS
- GCPR_CALLINGPARTY
- GCPR_MEDIADETECT
- GCPR_CALLPROGRESS

The **gc_GetParm( )** function, when used in this context, returns the information set using the **gc_SetParm( )** function. See Section 2.32, "gc_SetParm( )", on page 139 for more information on the meaning of these parameters.

2.14. gc_GetSigInfo( )

The **gc_GetSigInfo( )** function gets the signaling information from the 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 Section 5.4, "USRINFO_ELEM", on page 188. 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 the Appendix D, "DPNSS IEs and Message Types" and Appendix E, "DPNSS Call Scenarios".

2.15. gc_GetUserInfo( )

When using Springware boards, the **gc_GetUserInfo( )** function gets unprocessed information elements in the 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.

When using DM3 boards, the **gc_GetUserInfo( )** function is not supported.

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

Parameter	Input
target_type	GCTGT_GCLIB_NETIF, GCTGT_GCLIB_CHAN, GCTGT_GCLIB_CRN
target_id	need input

Parameter	Input
infoparmblkp	set ID - GCIS_SET_IE, parm ID - GCIS_PARM_UIEDATA

Termination Event

None

Cautions

- This function returns with an error if the Global Call application has set GCIS_PARM_RECEIVEINFOBUF. To retrieve unprocessed IEs, the application should use **gc_Extension**(GCIS_EXID_GETSIGINFO).
- This function is not supported for BRI/2 board.

Example

```
#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;
}
```

2.16. gc_HoldCall( )

The **gc_HoldCall**( ) function allows the application to place an active call on hold. For PRI protocols (including DPNSS and Q.SIG) and for BRI Network-side, the call must be in the Connected state to be put on hold. For BRI User-side, the call can be put on hold any time after the GCEV_PROCEEDING event is received.

NOTE: For other PRI protocols, the **gc_HoldCall**( ) function is not supported.

2.17. gc_MakeCall( )

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.

NOTE: For ISDN applications that use Springware boards, the maximum length of the **numberstr** parameter is 20 digits. If a **numberstr** value greater than 20 digits is specified, a GCEV_TASKFAIL event with an unknown ISDN error is received.

When using Springware boards, 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.

When using Springware boards, 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 1.9.1, "Set/Retrieve Configuration of a Logical Link (BRI Only)", on page 63 for more information. For PRI protocols, the only parameter 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( )**.

NOTE: For ISDN applications that use Springware boards, 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).

When using DM3 boards, the **timeout** parameter is supported in both synchronous and asynchronous modes.

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. The following paragraphs describe making an ISDN call from a D/xxxSC board and using the MAKECALL_BLK data structure in the **gc_MakeCall( )** function.

2.17.1. 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 (IE): 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 information elements (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.

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.

When using 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.

NOTES: **1.** You can set IEs in gclib and set cclib to NULL or you can set IEs in cclib and set gclib to NULL.

2. When using DM3 boards, if both the cclib and gclib pointers are set, the gclib pointer is ignored.

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 Section 5.3, "GC_MAKECALL_BLK", on page 181.

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. See Section 5.3, "GC_MAKECALL_BLK", on page 181.

Table 11 shows the parameters that can be included in the GC_MAKECALL_BLK associated with the **gc_MakeCall( )** function.

Table 11. 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
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

Table 11. Call Setup Parameters When Using gc_MakeCall( )

Parameter	Level	Description
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 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

† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.

Table 11. Call Setup Parameters When Using gc_MakeCall( )

Parameter	Level	Description
USRINFO_LAYER1_ PROTOCOL † (continued)	chan	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).
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

Table 11. Call Setup Parameters When Using gc_MakeCall( )

Parameter	Level	Description
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
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

The **gc_SetParm( )** function can also be used to specify call setup parameters. See Section 2.32, "gc_SetParm( )", on page 139 for more information.

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 first IE */
      ie_blk_ptr -> data [0]= 0x78;   /* IE type=78(TRANSIT NETWORK SELECTION) */
      ie_blk_ptr -> data [1]= 0x04;   /* the length of NOTIFY */
      ie_blk_ptr -> data [2]= 0xA1;   /* National network & carrier ID */

      /* The second IE  */
      ie_blk_ptr -> data [3]= 0x32;   /* IE type=28 (DISPLAY) */
      ie_blk_ptr -> data [4]= 0x38;   /* length of DISPLAY */
      ie_blk_ptr -> data [5]= 0x38;   /* caller name displayed */
   };

void main()
{
   int rc;
   char *devname = ":N_dtiB1T1:P_isdn";
   IE_BLK  ie_blk;

/* open the device */
```

```
   if (( rc = gc_Open(&lbuf, devname, 0)) < 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 for more information.

2.17.2. Restrictions When Using DM3 Boards

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 DM3 products provided with the system release software.

2. Neither the **gc_MakeCall( )** function nor the **gc_SetParm( )** function can be used to modify these parameters.

2.18. gc_OpenEx( )

The **gc_OpenEx( )** function is used to open both network board and channel (time slot) devices. This generic call control function initializes the specified time slot on the specified trunk. A line device ID will be returned to the application.

NOTE: When using Springware boards, **gc_OpenEx( )** does not support the inclusion of a voice device in the **devicename** string. Use the **dx_open( )** function to get a voice device handle.

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.

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.

The device to be opened is specified by the **devicename** parameter as defined in the **gc_OpenEx( )** function description in the *Global Call API Library Reference*. Both the protocol_name, which must be set to **isdn**, and network_device_name fields are required. The voice_device_name field is invalid when opening an ISDN device and should not be used. 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:

```
devicename = ":N_dtiB3T1:P_isdn";
```

- to open the board dtiB3, the **devicename** format is:

```
devicename = ":N_dtiB3:P_isdn";
```

Each PRI structure is composed of one D channel and 23 (T-1) or 30 (E-1) 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.

Cautions

- Do not open a D or B channel more than once from the same process, or you may get unpredictable results
- 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.

When using DM3 boards, 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 DM3 products provided with the system release software.

2.19. gc_ReleaseCallEx( )

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.

When using Springware boards, under PRI, 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 Figure 6, “Network Initiated Inbound Call (Synchronous Mode)”, on page 230, for more information on how to use this function.

When using 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.

Cautions

When using Springware boards using BRI protocols, under load conditions, or if the remote end delays transmitting the RELEASE COMPLETE message, an application could 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.

2.20. gc_ReqANI( )

When using DM3 boards, the **gc_ReqANI( )** function is not supported.

When using Springware 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's 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.

2.21. gc_ReqMoreInfo( )

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 *Global Call API Library Reference* for more information.

When using Springware boards, the **gc_ReqMoreInfo( )** is supported as documented in the *Global Call API Library Reference*.

2.22. gc_ReqService( )

This function is not supported for ISDN Technology.

2.23. gc_ResetLineDev( )

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.

2.24. gc_RespService( )

The **gc_RespService( )** function returns a response to a requested service. Refer to Section 1.10, "Service Request (BRI Only)", on page 84. Service Request for information of on how to use this function for ISDN applications.

When using DM3 boards, the **gc_RespService( )** function is not supported.

2.25. gc_RetrieveCall( )

The **gc_RetrieveCall( )** function is supported for DPNSS, Q.SIG and all BRI protocols.

NOTE: For other PRI protocols, the **gc_RetrieveCall( )** function is not supported.

2.26. gc_SetBilling( )

When using DM3 boards, the **gc_SetBilling( )** function is not supported.

When using 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 *Global Call API Software 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 12. These cause values apply only to AT&T's Vari-A-Bill service.

Table 12. Cause Values, 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.

Cautions

- This function is available only on the AT&T network and only for the PRI 4ESS protocol.

- The **gc_SetBilling( )** function may not function in all service-provider environments. Check whether retrieving billing information is an option with your service provider.

2.27. gc_SetCallingNum( )

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'.

Cautions

This function is not supported for the BRI/2 board.

2.28. gc_SetChanState( )

When power is initially applied, all bearer channels (B channels) are placed in the In Service state.

NOTE: In 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) and GCLS_OUT_OF_SERVICE (2).

NOTE: This feature may not be available in some countries.

Cautions

- E-1 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 E-1 ISDN protocols to avoid receiving incoming calls on some channels. An E-1 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.
- The **gc_SetChanState( )** function affects only the link between the calling process and the device. Other processes and devices are not affected.
- The **gc_SetChanState( )** is not supported for E-1 ISDN or NTT PRI protocols, or for any BRI protocols.

When using DM3 boards, 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 DM3 products provided with the system release software.

2.29. gc_SetConfigData( )

When using DM3 boards, the **gc_SetConfigData( )** function is not supported.

When using Springware boards, the **gc_SetConfigData( )** function supports the Global Call Run 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. Please refer to *Global Call API Library Reference* for detail on description of this API and for the ISDN parameters which are configurable using this API, please refer to Section 1.9, "Run Time Configuration Management", on page 62.

2.30. gc_SetEvtMsk( )

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.

NOTE: ISDN does not support disabling GCEV_BLOCKED or GCEV_UNBLOCKED in the **gc_SetEvtMsk( )** function

Table 13 shows the default values used by **gc_SetEvtMsk( )** for ISDN.

Table 13. bitmask Parameter Values

Parameter	Description	Default
GCMSK_NOFACILITYBUF Supported when using Springware boards; not supported when using DM3 boards.	Enable or disable for no facility buffer event	enabled
GCMSK_NOUSRINFO Supported when using Springware boards; not supported when using DM3 boards.	Enable or disable for no user information event	enabled
GCMSK_PROC_SEND Supported when using Springware boards; supported when using DM3 boards for board devices only.	Enable or disable for sending proceeding event	
GCMSK_PROGRESS Supported when using Springware and DM3 boards on line devices only and valid for the specified time slot	Enable or disable for sending progress event	
GCMSK_SETUP_ACK Supported when using Springware and DM3 boards on line devices only and valid for the specified time slot	Enable or disable for setup acknowledgement event	

2.31. gc_SetInfoElem( )

When using DM3 boards, the **gc_SetInfoElem( )** function is not supported. Use the **gc_SetInfoElem( )** function as an alternative.

The **gc_SetInfoElem( )** function allows the application to include application-specific ISDN IEs in the next outgoing message. The IE_BLK data structure used by this function to set additional IEs is defined in Section 5.1, "IE_BLK", on page 179.

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 the Appendix E, "DPNSS Call Scenarios" for more information.

NOTE: The **gc_SetInfoElem( )** can be used with any call control function except **gc_ReleaseCallEx( )**.

2.32. gc_SetParm( )

When using Springware boards, Table 14 lists the parameter selections that can be set using **gc_SetParm( )** function. All valid values are defined in the *isdncmd.h* header file. 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 14. 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
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

Table 14. Call Setup Parameters When Using gc_SetParm( )

Parameter	Level	Description
USRINFO_LAYER1_ PROTOCOL † (continued)	chan	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 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
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

Table 14. Call Setup Parameters When Using gc_SetParm( )

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).
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

Table 14. Call Setup Parameters When Using gc_SetParm( )

Parameter	Level	Description
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
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).
† Not supported on DM3 boards. Use the **gc_SetInfoElem( )** function instead.		

When using 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.

2.33. gc_SetUserInfo( )

The **gc_SetUserInfo( )** function is used to set additional Information elements, 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 Appendix D, "DPNSS IEs and Message Types", 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.

Cautions

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 **MAXLEN_IEDATA** of 254 bytes.

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;
}
```

2.34. gc_SndFrame( )

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 DM3 boards, the **gc_SndFrame( )** function returns an error if the number of bytes in the transmit frame exceeds the limit of 260 bytes.

2.35. gc_SndMoreInfo( )

When using Springware boards, the **gc_SendMoreInfo( )** function is supported as described in the *Global Call API Library Reference*.

When using 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.

2.36. gc_SndMsg( )

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. Possible message types are listed below. Messages specific to the DPNSS protocol are shown in bold; see also Appendix D, "DPNSS IEs and

Message Types" and Appendix E, "DPNSS Call Scenarios" for protocol-specific information.

- SndMsg_Congestion (supported on Springware and DM3 boards)
- **SndMsg_Divert** (supported on Springware boards only)
- SndMsg_Facility (supported on Springware and DM3 boards)
- SndMsg_FacilityACK (supported on Springware boards only)
- SndMsg_FacilityREJ (supported on Springware boards only)
- SndMsg_Information (supported on Springware and DM3 boards)
- **SndMsg_Intrude** (supported on Springware boards only)
- SndMsg_Notify (supported on Springware and DM3 boards)
- **SndMsg_NSI** (supported on Springware boards only)
- SndMsg_Status (supported on DM3 boards only)
- SndMsg_StatusEnquiry (supported on DM3 boards only)
- **SndMsg_Transfer** (supported on Springware boards only)
- **SndMsg_Transit** (supported on Springware boards only)
- SndMsg_UsrInformation (supported by Springware and DM3 boards)

2.37. gc_StartTrace( )

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.

This 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 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.

2.38. gc_StopTrace( )

The **gc_StopTrace( )** function discards any trace information stored in memory.

2.39. gc_WaitCall( )

A B channel (timeslot) 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_SetInfoElement( )** 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.

3. Sequence of Function Calls in ISDN

This chapter describes the sequence of function calls for the following ISDN call situations and modes:

- Inbound Calls, Asynchronous Mode
- Outbound Calls, Asynchronous Mode
- Call Termination, Asynchronous Mode
- Inbound Calls, Synchronous Mode
- Outbound Calls, Synchronous Mode
- Call Termination, Synchronous Mode

For more detailed ISDN call scenarios, see Appendix A, “Establishing ISDN Connections”.

3.1. Inbound Calls, Asynchronous Mode

Table 15 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode inbound call. The items denoted by an asterisk (*) are optional functions/events. To prevent interruptions by events that the application does not want to respond to, some events can be masked.

Table 15. ISDN Inbound Call Set-Up (Asynchronous)

Function/Event	Action/Description
gc_WaitCall( )	Issued once after line device opened with **gc_Open( )** or **gc_OpenEx( )** Incoming calls are unblocked
* = Optional function/event	

Table 15. ISDN Inbound Call Set-Up (Asynchronous) (Continued)

Function/Event	Action/Description
GCEV_OFFERED	Indicates arrival of an incoming call; A Setup message was received from the network; Proceeding message sent to network; • When using Springware boards, by default, the Proceeding message is automatically sent to the network • When using 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 buffer.
gc_GetANI( )* (when using Springware or DM3 boards) OR **gc_ReqANI( )*** (when using Springware boards only)	Information returned is stored in buffer. Request ANI information.
gc_CallProgress( )* (when using Springware boards only)	Progress message sent to acknowledge that the call was received. No response expected from network.
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.

* = Optional function/event

Table 15. ISDN Inbound Call Set-Up (Asynchronous) (Continued)

Function/Event	Action/Description
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.
* = Optional function/event	

3.2. Outbound Calls, Asynchronous Mode

Table 16 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode outbound call. The items denoted by an asterisk (*****) are optional events that may be reported to the application for specific signaling protocols.

Table 16. ISDN Outbound Call (Asynchronous)

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.
* = Optional event	

Table 16. ISDN Outbound Call (Asynchronous)

Function/Event	Action/Description
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( )**.
* = Optional event	

3.3. Call Termination, Asynchronous Mode

Table 17 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during an asynchronous mode call termination.

Table 17. Call Termination (Asynchronous)

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.

Table 17. Call Termination (Asynchronous)

Function/Event	Action/Description
gc_ReleaseCall( )	Issued to release all resources used for call; network port is ready to receive next call. Causes transition to Null state.

3.4. Inbound Calls, Synchronous Mode

Table 18 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode inbound call. The items denoted by an asterisk (*) are optional functions/events or maskable events that may be reported to the application for specific signaling protocols.

Table 18. ISDN Inbound Call Set-Up (Synchronous)

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 Springware boards, by default, the Proceeding message is automatically sent to the network • When using DM3 boards, by default, the application must explicitly use the **gc_CallAck( )** function to send the Proceeding message **Note:** 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.
gc_GetANI( )*	Information returned is stored in buffer.
* = Optional function/event	

Table 18. ISDN Inbound Call Set-Up (Synchronous)

Function/Event	Action/Description
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( )	**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.
* = Optional function/event	

3.5. Outbound Calls, Synchronous Mode

See Table 19 for the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode outbound call. The items denoted by an asterisk (*****) 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 *Global Call API Library Reference* for a list of maskable events.

Table 19. ISDN Outbound Call (Synchronous)

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.
* = Optional event	

3.6. Call Termination, Synchronous Mode

Table 20 describes the sequencing of function calls and the messages exchanged with the ISDN carrier during a synchronous mode call termination.

Table 20. Call Termination (Synchronous)

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.

4. ISDN-Specific Parameter Set Reference

This chapter describes the ISDN-specific parameter sets and parameter IDs that are defined in the *gcisdn.h* header file.

NOTE: The parameter set IDs and parameter IDs described in this chapter apply to Springware boards only. When using DM3 boards these parameter set IDs and parameter IDs are **not** supported.

The parameter sets include:

- GCIS_SET_ADDRESS Parameter Set
- GCIS_SET_BEARERCHNL Parameter Set
- GCIS_SET_CALLPROGRESS Parameter Set
- GCIS_SET_CHANSTATE Parameter Set
- GCIS_SET_DCHANCFG Parameter Set
- GCIS_SET_DLINK Parameter Set
- GCIS_SET_DLINKCFG Parameter Set
- GCIS_SET_EVENTMSK Parameter Set
- GCIS_SET_FACILITY Parameter Set
- GCIS_SET_GENERIC Parameter Set
- GCIS_SET_IE Parameter Set
- GCIS_SET_SERVREQ Parameter Set
- GCIS_SET_SNDMSG Parameter Set
- GCIS_SET_TONE Parameter Set

4.1. GCIS_SET_ADDRESS Parameter Set

Table 21 shows the parameter IDs for the GCIS_SET_ADDRESS set ID.

NOTE: The GCIS_SET_ADDRESS parameter set is not supported when using DM3 boards. Use **gc_MakeCall( )** to set and **gc_GetSigInfo( )** to retrieve called number and calling number information.

Table 21. 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.)

Table 21. GCIS_SET_ADDRESS Parameter IDs (Continued)

Parameter ID	Type	Description
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.)

4.2. GCIS_SET_BEARERCHNL Parameter Set

Table 22 shows the parameter IDs for the GCIS_SET_BEARERCHNL set ID.

NOTE: The GCIS_SET_BEARERCHNL parameter set is not supported when using DM3 boards. 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 22. GCIS_SET_BEARERCHNL Parameter IDs

Parameter ID	Type	Description
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 Note: If this parameter is not present in the makecall block then CCLIB will use the value set through RTCM.

4.3. GCIS_SET_CALLPROGRESS Parameter Set

Table 23 shows the parameter IDs for the GCIS_SET_CALLPROGRESS set ID.

NOTE: The GCIS_SET_CALLPROGRESS parameter set is not supported when using DM3 boards.

Table 23. 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

4.4. GCIS_SET_CHANSTATE Parameter Set

Table 24 shows the parameter IDs for the GCIS_SET_CHANSTATE set ID.

NOTE: The GCIS_SET_CHANSTATE parameter set is not supported when using DM3 boards.

Table 24. 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 in operable. 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.

4.5. GCIS_SET_DCHANCFG Parameter Set

The parameter set is used for 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.

NOTE: The GCIS_SET_DCHANCFG parameter set is not supported when using DM3 boards.

When the switch is operating as the User side in North American protocols, the **gc_SetConfigData( )** (to set DSL configuration) function 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( )** (to set DSL configuration) function 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( )** (to set DSL configuration) function 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 25. 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. where: • 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

Table 25. GCIS_SET_DCHANCFG Parameter IDs (Continued)

Parameter ID	Type	Description
		• 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 1 to MAX_DLINK range, where MAX_DLINK is currently set to 8. This field only has significance when configuring the DSL as the NETWORK side.

Table 25. GCIS_SET_DCHANCFG Parameter IDs (Continued)

Parameter ID	Type	Description
GCIS_PARM_DCHANCFG_SPID	char	(ASCII digit string limited to the digits 0-9 and limited in length to MAX_SPID_SIZE). – Defines the assigned Service Provider Identifier (SPID) value for terminal initialization. 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.
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

Table 25. GCIS_SET_DCHANCFG Parameter IDs (Continued)

Parameter ID	Type	Description
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 Example: typedef enum { ISDN_INVALID_SWITCH=0x80, ISDN_BRI_5ESS, ISDN_BRI_DMS100, ISDN_BRI_NTT, ISDN_BRI_NET3, ISDN_BRI_NI1, ISDN_BRI_NI2 } IsdnSwitchType;

Table 25. GCIS_SET_DCHANCFG Parameter IDs (Continued)

Parameter ID	Type	Description
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 (see below). 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.

4.6. GCIS_SET_DLINK Parameter Set

Table 26 shows the parameter IDs for the GCIS_SET_DLINK set ID.

NOTE: The GCIS_SET_DLINK parameter set is not supported when using DM3 boards.

Table 26. GCIS_SET_DLINK Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_DLINK_CES	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	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	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 in operable. 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.

4.7. GCIS_SET_DLINKCFG Parameter Set

The GCIS_SET_DLINKCFG set ID encapsulates parameters required to initialize the firmware structures to allow the logical link to be used. Table 27 shows the parameter IDs for the GCIS_SET_DLINKCFG set ID.

NOTE: The GCIS_SET_DLINKCFG parameter set is not supported when using DM3 boards.

Table 27. 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 in operable. 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.

Table 27. GCIS_SET_DLINKCFG Parameter IDs (Continued)

Parameter ID	Type	Description
GCIS_PARM_DLINKCFG_TEI	char	Terminal Endpoint Identifier. Valid values are: • 0 - 63 – for manual TEIs (chosen by the user side) • AUTO_TEI – for automatic TEIs (chosen by the network side)

4.8. GCIS_SET_EVENTMSK Parameter Set

Table 28 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.

NOTE: The GCIS_SET_EVENTMSK parameter set is not supported when using DM3 boards. See Section 2.30, "gc_SetEvtMsk( )", on page 138 for more information on masking events on DM3 boards.

.

Table 28. 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_TERMINATE – Delay RELEASE COMPLETE message until host application calls the **gc_ReleaseCall( )** function. Default: Firmware does not wait for the gc_ReleaseCall() function and sends RELEASE COMPLETE when RELEASE is received.

Table 28. GCIS_SET_EVENTMSK Parameter IDs (Continued)

Parameter ID	Type	Description
		• 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_PROC_SEND – Receiving the GCEV_EXTENSION event when the information is successfully sent. 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.

4.9. GCIS_SET_FACILITY Parameter Set

Table 29 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.

NOTE: The GCIS_SET_FACILITY parameter set is not supported when using DM3 boards.

Table 29. 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.

4.10. GCIS_SET_GENERIC Parameter Set

Table 30 shows the parameter IDs for the GCIS_SET_GENERIC set ID.

NOTE: The GCIS_SET_GENERIC parameter set is not supported when using 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 30. 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.

Table 30. GCIS_SET_GENERIC Parameter IDs (Continued)

Parameter ID	Type	Description
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 value lies 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).

4.11. GCIS_SET_IE Parameter Set

Table 31 shows the parameter IDs for the GCIS_SET_IE set ID.

NOTE: The GCIS_SET_IE parameter set is not supported when using DM3 boards. Use **gc_SetParm( )** to set the calling multiple IE buffer size. The calling multiple IE buffer size cannot be retrieved.

Table 31. 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 • gcis_SetUserInfo.

Table 31. GCIS_SET_IE Parameter IDs (Continued)

Parameter ID	Type	Description
GCIS_PARM_UIEDATA	char array, length should not exceed GCIS_MAXLEN_ IEDATA	This parameter is used to pass unprocessed IEs from CCLIB to APP.

4.12. GCIS_SET_SERVREQ Parameter Set

Table 32 shows the parameter IDs for the GCIS_SET_SERVREQ set ID.

NOTE: The GCIS_SET_SERVREQ parameter set is not supported when using DM3 boards.

Table 32. GCIS_SET_SERVREQ Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_SERVREQ_CAUSE VALUE	unsigned char	Valid values are: • NETWORK_OUT_OF_ORDER • BAD_INFO_ELEM • INVALID_ELEM_CONTENTS • TIMER_EXPIRYPROTOCOL_ERROR For a description of cause values, see Table 33.
GCIS_PARM_SERVREQ_INTER PRETER	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 – 63. 00 signifies that the firmware is to provide a default.
GCIS_PARM_SERVREQ_USID	unsigned char	User Service Identifier. The range is 01 – FF. 00 signifies default.

Table 33. Cause 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.

4.13. GCIS_SET_SNDMSG Parameter Set

Table 34 shows the parameter ID for the GCIS_SET_SNDMSG set ID.

NOTE: The GCIS_SET_SNDMSG parameter set is not supported when using DM3 boards.

Table 34. GCIS_SET_SNDMSG Parameter IDs

Parameter ID	Type	Description
GCIS_PARM_SNDMSGTYPE	int	Valid ISDN message types are: All protocols: • SndMsg_Information • SndMsg_Congestion • SndMsg_UsrInformation • SndMsg_Facility • SndMsg_FacilityACK • SndMsg_FacilityREJ • SndMsg_Notify • SndMsg_ServiceAck • SndMsg_Status • SndMsg_StatusEnquiry • SndMsg_GlobalStatus

4.14. GCIS_SET_TONE Parameter Set

Table 35 shows the parameter IDs for the GCIS_SET_TONE set ID.

NOTE: The GCIS_SET_TONE parameter set is not supported when using DM3 boards.

Table 35. 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_TERMPARM LENGTH	unsigned short	Duration for which tone has to be played.

5. Data Structure Reference

The following data structures are used for ISDN applications and are described in this chapter:

- IE_BLK
- L2_BLK
- GC_MAKECALL_BLK
- USRINFO_ELEM
- DLINK
- DLINK_CFG
- NONCRN_BLK
- SPID_BLK
- TERM_BLK
- TERM_NACK_BLK
- ToneParm
- USPID_BLK
- DCHAN_CFG

The data structure definition is followed by a table, which provides a detailed description of the parameters and data elements in the data structure and possible values. These parameters and data elements are listed in the sequence in which they are defined in the data structure. Refer to the *Global Call API Library Reference* for additional data structures used by Global Call.

5.1. IE_BLK

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 *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. The IE_BLK data structure is defined as follows:

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

See Table 36 for descriptions of the IE_BLK fields.

Table 36. IE_BLK Fields

Field	Description
length	Length of data block in bytes. Value must be less than MAXLEN_IEDATA as defined in *gcisdn.h*.
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.

5.2. L2_BLK

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 the **gc_GetFrame( )** function. See example code for these functions in the *Global Call API Library Reference* for details.

The L2_BLK data structure is defined as follows:

```
#define MAXLEN_IEDATA    254
typedef struct
  {
   char sapi;
   char ces;
   short length;
   char data[MAXLEN_DATA];
   } L2_BLK, *L2_BLK_PTR;
```

See Table 37 for descriptions of the L2_BLK fields.

Table 37. L2_BLK Fields

Field	Description
sapi	service access point ID (always set to 0)
ces	connection endpoint suffix • When using Springware boards: set to 0 • When using DM3 boards: set to 1
length	Length of data block in bytes. Value must be less than MAXLEN_IEDATA as defined in *gcisdn.h*.
data[MAXLEN_IEDATA]	Data for frame. Must be formatted to meet CCITT recommendations. The maximum length of the data field is MAXLEN_IEDATA.

The following 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 */
```

5.3. GC_MAKECALL_BLK

The GC_MAKECALL_BLK structure contains information used by the **gc_MakeCall( )** function when setting up a call. The structure is defined as follows:

```
typedef struct
{
   GCLIB_MAKECALL_BLK  *gclib;
   void                *cclib;
} GC_MAKECALL_BLK, *GC_MAKECALL_BLKP;
```

The gclib pointer 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 description of the fields in the GCLIB_MAKECALL_BLK structure, refer to the *Global Call API Library Reference*.

There are certain parameters, which are defined for ISDN only. These parameters can be defined in the ext_datap GC_PARM_BLK. Table 38 list the parameters that can be included.

Table 38. ISDN Parameters

set ID	Parameter ID	Description	Supported Values
GCIS_SET_ BEARERCHNL (for Springware boards) ISDN_SET_BEARER_ CHNL (for DM3 boards)	GCIS_PARM_ TRANSFERMODE (for Springware boards) ISDN_TRANSFER_ MODE (for DM3 boards)	Bearer Channel information transfer mode	ISDN_ITM_CIRCUIT – circuit switch mode
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 38. ISDN Parameters

set ID	Parameter ID	Description	Supported Values
GCIS_SET_ BEARERCHNL (for Springware boards) ISDN_SET_BEARER_ CHNL (for DM3 boards)	GCIS_PARM_ TRANSFER_RATE (for Springware boards) ISDN_TRANSFER_ RATE (for DM3 boards)	Bearer Channel information transfer rate	BEARER_RATE_64KBPS – 64 K bps transfer rate
ISDN_SET_ CALL_INFO	ISDN_INFO_ ELEMENTS	User Information element	value_buf field of GC_PARM_DATA contains a pointer to IE_BLK (refer to section 5.1. IE_BLK for description of IE_BLK) and contains the information element to be sent to the network.
GCIS_SET_ FACILITY (for Springware boards) ISDN_SET_ FACILITY (for DM3 boards)	GCIS_PARM_ FACILITY_ FEATURESERVICE (for Springware boards) ISDN_FACILITY_ FEATURE_SERVICE (for DM3 boards)	Identifies facility request as a feature or a service (See Note, below)	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.
GCIS_SET_ FACILITY (for Springware boards) ISDN_SET_ FACILITY (for DM3 boards)	GCIS_PARM_ FACILITY_ CODINGVALUE (for DM3 boards) ISDN_FACILITY_ CODING_VALUE (for DM3 boards)	Facility coding value; identifies the specific feature or service provided (See Note, below)	ISDN_CPN_PREF – calling party number preferred ISDN_SDN – AT&T Software Defined Network ISDN_BN_PREF – Billing Number preferred
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.			

When using Springware boards, a sample MAKECALL_BLK initialization is shown below:

```
#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);
     }
}
```

5.3.1. Setting Information Elements

The information elements (IEs) sent to the network must conform to the switch-specific recommendations. Use the assumptions described in the following paragraphs when constructing IEs. See also Section 2.31, “gc_SetInfoElem( )”, on page 139.

Assumption 1

Variable length IEs must be provided in ascending order in the Public part, as shown in Table 39.

Table 39. Variable Length IEs

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 40 for Type 1 settings and Table 41 for Type 2 settings.

Table 40. 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 41. 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 42 for two options for setting LOCKING Shift IEs.

Table 42. 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 43. 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 43. LOCKING Shift IEs - Option 2

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 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.

5.4. USRINFO_ELEM

The USRINFO_ELEM data structure is used to return User-to-User Information (UUI) data using the **gc_GetCallInfo( )** or the **gc_GetSigInfo( )** function. The USRINFO_ELEM data structure is defined as follows:

```
typedef struct {
   unsigned char length;    /* protocol_discriminator + user information
                               length */
   unsigned char protocol_discriminator;
   char usrinformation[256];
} USRINFO_ELEM, *USRINFO_ELEM_PTR;
```

Table 44. USRINFO_ELEM Fields

Field	Description
length	First byte 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	Second byte defines the network protocol.
usrinformation	Data containing the application dependent user information.

5.5. DLINK

When using DM3 boards, the DLINK data structure is not supported.

When using Springware boards, the DLINK data structure contains information about the data link information block.

The DLINK structure is used in the following structures:

- SPID_BLK
- TERM_BLK
- TERM_NACK_BLK
- USPID_BLK

The DLINK structure is defined as follows:

```
typedef   struct
{
    char  sapi;
    char  ces;
}  DLINK,   *DLINK_PTR;
```

Table 45. DLINK Field Descriptions

Field	Description
sapi	The service access pointer identifier. (This field is zero for PRI.)
sapi	The service access pointer identifier. (This field is zero for PRI.)

5.6. DLINK_CFG

When using DM3 boards, the DLINK_CFG data structure is not supported.

When using Springware boards, the DLINK_CFG structure contains information about the data link logical link configuration block.

The structure is defined as follows:

```
typedef   struct
{
    char  tei;
    int   state;
    int   protocol;
}  DLINK_CFG,   *DLINK_CFG_PTR;
```

Table 46. DLINK_CFG Field Descriptions

Field	Description
tei	Terminal Endpoint Identifier. Valid values are: • 0 - 63 - for manual TEIs (chosen by the user side) • AUTO_TEI - for automatic TEIs (chosen by the network side)

Table 46. DLINK_CFG Field Descriptions

Field	Description
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 and 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 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.

5.7. NONCRN_BLK

When using DM3 boards, the NONCRN_BLK data structure is not supported.

When using Springware boards, the NONCRN_BLK structure contains information related to a GLOBAL or NULL call reference number (CRN). The messages are sent to the network using the **SndNonCallMsg( )** function.

The structure is defined as follows:

```
#define MAXLEN_IEDATA  254
typedef struct
  {
    char    sapi;
    char    ces;
    short   length;
    char  data[MAXLEN_IEDATA];
  } NONCRN_BLK, *NONCRN_BLK_PTR;
```

Table 47. NONCRN_BLK Field Descriptions

Field	Description
sapi	The Service Access Point Identifier. For call control procedures, this value is always zero.
ces	The Service Access Point Identifier. For call control procedures, this value is always zero.
length	The total bytes in the data field
data	This field contains the information element(s) to be sent.

5.8. SPID_BLK

When using DM3 boards, the SPID_BLK data structure is not supported.

When using Springware boards, the SPID_BLK data structure is used to cast terminal initialization event data after a CCEV_TERM_REGISTER event is received. SPID_BLK contains the value of the Service Profile Interface ID, which is used to determine whether the value is valid for a designated service.

The data structure is defined as follows:

```
/*
 * Structure for CCEV_TERM_REGISTER Event.
 */
typedef struct
{
   DLINK data_link;
   byte  initializing_term;
   byte  SPID[MAX_SPID_SIZE];
} SPID_BLK;
```

Table 48. SPID_BLK Field Descriptions

Field	Description
data_link	The DLINK data structure; see Section 5.5, “DLINK”, on page 189.
initializing_term	The type of initializing terminal.
SPID	The Service Profile Interface ID.

5.9. TERM_BLK

When using DM3 boards, the TERM_BLK data structure is not supported.

When using Springware boards, the TERM_BLK data structure contains information regarding the application's response to the GCEV_SERVICERESP event. The response is sent using the **GCEV_SERVICERESP( )** function.

The structure is defined as follows:

```
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;

/* where DATA_LINK contains the following structure */

typedef struct
{
   byte  sapi;
   byte  ces;
} DLINK, *DLINK_PTR;
```

Table 49. TERM_BLK Field Descriptions

Field	Description
data_link	The DLINK data structure; see Section 5.5, "DLINK", on page 189
ack_type	The type of acknowledgement to be passed to the firmware. The settings are: • ISDN_OK - to send a Positive acknowledgment • ISDN_ERROR - to send a Negative acknowledgment
cause_value	The Cause Value defined in *isdncmd.h*. (For a listing of Cause Values, see Table 33, "Cause Values", on page 176.)
usid	User Service Identifier. The range is 01 – FF. 00 signifies default.
tid	Terminal Identifier. The range is 01 – 63. 00 signifies that the firmware is to provide a default.
interpreter	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

5.10. TERM_NACK_BLK

When using DM3 boards, the TERM_NACK_BLK data structure is not supported.

When using Springware boards, the TERM_NACK_BLK data structure is used to cast terminal initialization event data after a CCEV_RCVTERMREG_NACK event is received. TERM_NACK_BLK contains the cause value for the event, indicating why the terminal initialization request was rejected by the network.

The data structure is defined as follows:

```
/*
 * Structure for CCEV_RCVTERMREG_NACK Event.
 */
typedef struct
{
   DLINK data_link;
   byte  cause_value;
} TERM_NACK_BLK;
```

Table 50. TERM_NACK_BLK Field Descriptions

Field	Description
data_link	The DLINK data structure; see Section 5.5, “DLINK”, on page 189.
cause_value	The Cause Value defined in *isdncmd.h*. (For a listing of Cause Values associated with a CCEV_RCTERMREG_NACK event. See Table 51 below.

The following table 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 will also be passed up to the application.

Table 51. 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.

Table 51. Cause Values Associated with CCEV_RCVTERMREG_NACK

Cause Value	Q.850 Description	Meaning
0x6F	Protocol error, unspecified	Used when no cause was given for the rejection.

5.11. ToneParm

When using DM3 boards, the ToneParm data structure is not supported.

When using Springware 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: Global Call 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 *ISDN Software Reference*, in which these functions are documented, may not be included in the documentation for future system releases.

The data structure is defined as follows:

```
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
}
```

Table 52. ToneParm Field Descriptions

Field	Description
duration	Specifies the duration of the tone in 10 ms units. The range is 1 - 65535. Set to -1 to play forever.
freq1	Specifies the frequency of the tone. The range is 200 - 3100 Hz.
amp1	Specifies the amplitude of the tone. The range is -40 - +3 dB.
freq2	Specifies the frequency of the tone. The range is 200-3100 Hz.
amp2	Specifies the amplitude of the tone. The range is -40 - +3 dB.
toneOn1	Specifies the tone interval, in 10 ms units. The range is 1 - 65535 ms. Set to 1 or greater for continuous tone.
toneOff1	Specifies the tone interval, in 10 ms units. The range is 0 - 65534 ms. Set to 0 to play a continuous tone.
reserv1	Reserved for future use
reserv2	Reserved for future use

5.12. USPID_BLK

When using DM3 boards, the USPID_BLK data structure is not supported.

When using Springware boards, the USPID_BLK data structure is used to cast terminal initialization event data after a CCEV_RCVTERMREG_ACK event is received. USPID_BLK contains the value of a valid User Service Profile Interface.

The data structure is defined as follows:

```
/* Structure for CCEV_RCVTERMREG_ACK Event /
typedef struct
{
   DLINK data_link;
   struct
   {
```

```
      byte usid;
      byte tid;
      byte interpreter;
   } uspid;
} USPID_BLK;
```

Table 53. USPID_BLK Field Descriptions

Field	Description
data_link	The DLINK data structure; see Section 5.5, "DLINK", on page 189.
usid	User Service Identifier. The range is 01 – FF. 00 signifies default.
tid	Terminal Identifier. The range is 01 – 63. 00 signifies that the firmware is to provide a default.
interpreter	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

5.13. DCHAN_CFG

When using DM3 boards, the DCHAN_CFG data structure is not supported.

When using Springware 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.

The structure is defined as follows:

```
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;
```

The possible values for the DCHAN_CFG structure are listed below. All components of the DCHAN_CFG structure that pertain to your configuration must be set. There are no default values.

NOTE: The T3xx values for defining the Layer 3 timer can be used for BRI/2, BRI/SC and PRI protocols. All of the other values in the structure are applicable only to BRI/SC.

Table 54. DCHAN_CFG Field Descriptions and Values

Type	Description	Values

layer2_access	Boolean value used to configure the DSL for direct layer 2 access or for full stack access.	`#define LAYER_2_ONLY 0`
		`#define FULL_ISDN_STACK 1`
		Where:·
		• LAYER_2_ONLY = ISDN access at layer 2 (If LAYER_2_ONLY is selected, no other parameters are required)·
		• FULL_ISDN_STACK = ISDN access at L3 call control

switch_type	Basic rate protocol (switch type) for DSL. Multiple run-time selectable switch types are available.	`typedef enum { ISDN_INVALID_SWITCH=0x80, ISDN_BRI_5ESS, ISDN_BRI_DMS100, ISDN_BRI_NTT, ISDN_BRI_NET3, ISDN_BRI_NI1, ISDN_BRI_NI2 } IsdnSwitchType;` Where: • 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

Table 54. DCHAN_CFG Field Descriptions and Values (Continued)

Type	Description	Values

switch_side	Boolean value defining whether the DSL should be configured as the Network side (NT) or the User side (TE).	`#define USER_SIDE 0`
		`#define NETWORK_SIDE 1`
		Where:
		• USER_SIDE = User side of ISDN protocol
		• NETWORK_SIDE = Network side of ISDN protocol

number_of_ endpoints	Number of logical data links to be supported.	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.

Table 54. DCHAN_CFG Field Descriptions and Values (Continued)

Type	Description	Values
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. `#define NO_PCM_TONE 0x00#define ULAW_PCM_TONE 0x01#define ALAW_PCM_TONE 0x02#define DEFAULT_PCM_TONE 0x03#define SENDING_COMPLETE_ATTACH 0x04#define USER_PERST_L2_ACT 0x08#define HOST_CONTROLLED_RELEASE 0x10` Where: • 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( )**
feature_controlB	Firmware feature control field. This is a bit mask field for setting features in the firmware.	Currently not used.

Table 54. DCHAN_CFG Field Descriptions and Values (Continued)

Type	Description	Values
rfu_1 & rfu_2	Reserved for future use.	Currently not used.

tei_assignment	Applies to User Side only. It specifies if the terminal has a fixed TEI or an auto-assigning TEI. If it is fixed, then "fixed_tei_value" must be specified (see below).	`#define AUTO_TEI_TERMINAL 0`
		`#define FIXED_TEI_TERMINAL 1`
		Where:
		• AUTO_TEI_TERMINAL = auto TEI assigning Term
		• FIXED_TEI_TERMINAL = Fixed TEI assigning Term

fixed_tei_value	Defines the TEI to be used for a fixed TEI assigning terminal.	0 to 63 (Required when tei_assignment = FIXED_TEI_TERMINAL)

auto_init_flag	Boolean value defining whether or not 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.	`#define AUTO_INIT_TERMINAL 0`
		`#define NON_INIT_TERMINAL 1`
		Where:
		• AUTO_INIT_TERMINAL = auto initializing terminal
		• NON_INIT_TERMINAL = non-auto initializing term

SPID	Defines the assigned Service Provider Identifier (SPID) value for terminal initialization. It is only applicable to User side US switches. When you set the SPID, it is assigned to both bearer channels associated with the D channel.	ASCII digit string limited to the digits 0-9 and limited in length to MAX_SPID_SIZE Where: MAX_SPID_SIZE = (20+1) (Required when auto_init_flag = AUTO_INIT_TERMINAL. Most North American switches require a SPID.)
no_am.rfu_1 & rfu_2	Reserved for future use.	Currently not used.

Table 54. DCHAN_CFG Field Descriptions and Values (Continued)

Type	Description	Values
rfu[RFU_COUNT]	Array of fields reserved for future use.	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.
T3xx (T302, T303, T304, T305, T306, T308, T309, T310, T312, T313, T318, T319, T322)	Defines the Layer 3 timer 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.	

6. Protocols

Like any evolving technology, ISDN implementation has yet to be agreed upon worldwide. Standards have been established in a number of countries or regions. The standards supported by Global Call products are described in this chapter.

The protocols supported, the firmware and parameter files for each protocol and the protocol parameters are described in the following paragraphs.

6.1. Basic Rate Interface

There are two types of Global Call BRI boards, BRI/SC and BRI/2:

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

NOTES: 1. For information on using Voice Functions, see the *Voice API Programming Guide* and *Voice API Library Reference*.

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

The Global Call BRI/SC and BRI/2 boards provide network access via the ISDN Basic Rate Interface (BRI). The 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 BRI/SC boards may also be used for connecting voice processing applications to PBX or Public Switched Telephone Network (PSTN) BRI access lines.

6.1.1. 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 BRI/SC or BRI/2 board can be used to connect to a PBX.

Both the 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 Chapter 2, "Applying Global Call Functions to ISDN Applications", 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( )**, **gc_SndNonCallMsg( )**, and **gc_SetInfoElem( )** functions for information on sending Facility Messages. See the **gc_GetCallInfo( )** and **gc_GetNonCallMsg( )** functions for information on retrieving Facility Messages. Also refer to Appendix C, "BRI Supplemental Services".

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

- Fax features - 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 (4 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 the BRI/2, see the *Fax Software Reference for Windows*.

- Data features - BRI/2 boards provide link layer access, across the B channel, which allows for reliable transfer of data across an ISDN network. The 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 BRI/2 by providing a standard interface. This means that the BRI/2 may be used by applications that use the standard networking APIs that are part of the Windows operating system. NDIS supports Remote Access Service (RAS) and Point-to-Point Protocol (PPP):
- 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.

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

- Data Link Layer Access - the BRI/SC products 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 a Global Call

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 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 *ISDN Software Reference*, in which these functions are documented, may not be included in the documentation for future 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. See the **gc_SetDChanCfg( )** function description in Chapter 2, "Applying Global Call Functions to ISDN Applications", for more information.
- 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. See the **gc_SndMsg( )** function description in Chapter 2, "Applying Global Call Functions to ISDN Applications", for more information.

6.1.2. 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

6.2. ISDN Primary Rate Interface

The Global Call ISDN Primary Rate Interface (PRI) firmware supports both T-1 and E-1 protocols.

The T-1 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 E-1 protocol implementations comply with the E-1 ISDN PRI protocols. The E-1 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.

6.3. Protocols Supported

A standard ISDN interface providing 23 (T-1) or 30 (E-1) voice or data channels (B channels) and one signaling channel (D channel) is available. For a list of supported protocols, you can:

- check the Release Catalog for your system software

- contact your nearest sales office at *http://www.intel.com/network/csp/sales*
- visit the support website at *http://developer.intel.com/design/telecom/support/*

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 (when using Springware boards) or in the CONFIG file (when using DM3 boards) 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.

6.4. User Configurable ISDN Parameters

When using DM3 boards, ISDN protocol parameters are configurable by editing a CONFIG file. The FCDGEN utility is then used to convert the CONFIG file to an FCD file which is then used when downloading the firmware to the board. See the configuration information for DM3 products provided with the system release software for more information about the parameters that can be configured.

When using Springware boards, the parameters listed in Table 55 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 55. Modifiable Protocol Parameters

Parameter	Value (hex)	Description
000F	00* 01	Digital E-1 trunk framing format: • 00 = G.703 framing without CRC4 (default) • 01 = G.703 framing with CRC4 Must be set to carrier requirement. **Note:** For T-1 applications, this parameter must be commented out (not used).
0013	00* 01	Digital trunk protocol type: • 00 = Standard T-1/E-1 (default) • 01 = PRI ISDN Must be set to 01 for ISDN application.
0014	00* 01	Digital T-1 trunk framing format: • 00 = D4 framing (default) • 01 = ESF (Extended Super Frame) framing **Note:** ESF framing is only supported in SCbus mode. **Note:** For E-1 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 55. Modifiable Protocol Parameters (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 **Note:** 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 56 and Table 57 for a summary of the default parameter settings for each protocol.

Table 56. T-1 ISDN Protocol Parameter Defaults

Parameter	4ESS, 5ESS	DMS/100, DMS/250	NTT (INS1500)
000F	----	----	----
0013	01	01	01
0014	Check with carrier.		

Table 56. T-1 ISDN Protocol Parameter Defaults (Continued)

Parameter	4ESS, 5ESS	DMS/100, DMS/250	NTT (INS1500)
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 55.		

Table 57. E-1 ISDN Protocol Parameter Defaults

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 55.					

6.5. Protocol Components

When using 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 Springware boards, each protocol requires specific firmware parameter file(s) to be downloaded to the network boards.

6.6. Using ISDN Protocols with Global Call System Software

When using DM3 boards, select an ISDN protocol by choosing the appropriate FCD file at board configuration time:

- Linux: The ISDN protocol to be used by a board is specified by selecting which Feature Configuration Description (FCD) file to use for each board in the System Configuration Description (SCD) file.
- Windows: The ISDN protocol to be used by a board is specified by choosing a Feature Configuration Description (FCD) file to use for each board in the Intel® Dialogic Configuration Manager (DCM).

See the configuration information for DM3 products provided with the system release sotware for more information.

When using 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 E-1 ISDN protocol:

```
ISDNProtocol = ctr4
ParameterFile = isdnE1.prm
```

- Windows: Use the Intel® Dialogic Configuration Manager utility to select the protocol for each board.

7. Debugging Applications

This chapter describes the Global Call Diagnostic utilities used to test and debug 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.

The diagnostic utilities included in the Global Call API ISDN software package comprise:

- ISDN Network Firmware (NT1 and NE1) - to provide network side emulation
- ISDN Diagnostic Program (*isdiag*) - to initiate calls, alter call set-up parameters, and initiate traces of the D channel activity on the trunk
- ISDTRACE Utility (*isdtrace*) - to easily convert the binary trace files (*filename.log*) of the D channel communications into a formatted text file (*filename.res*) that is easy to read and analyze. The binary trace file is generated using the **gc_StartTrace( )** and **gc_StopTrace( )** functions.

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

7.1. ISDN Network Firmware

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 Intel® 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.

To use this utility:

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. Setup 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 Intel® Dialogic® configuration file for the network board used to emulate the ISDN network. For Windows applications, use the Intel® Dialogic Configuration Manager (DCM) utility to make these protocol changes.
   - setting the configurable ISDN network parameters to match those used in your application; see Table 55, "Modifiable Protocol Parameters", on page 211.
   - resetting this host PC to load the emulation firmware and the ISDN network emulation parameters.

4. Run your application.

7.2. ISDN Diagnostic Program

When using 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.

NOTE: The ISDN Diagnostic program (*isdiag*) is **not** supported when using DM3 boards.

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 (T-1 or E-1) 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 T-1 and e for E-1)
   - 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 Q.SIG protocols only)

- set supplementary DPNSS/Q.SIG services (intrusion, local diversion, remote diversion, virtual calls for inbound/outbound) (DPNSS and Q.SIG protocols only)

ESC exit

F1 help menu; describes the main menu options

7.3. ISDTRACE Utility

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 selects 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
```

Appendix A:
Establishing ISDN Connections

Pointers for ordering ISDN Primary Rate service and establishing a connection between the Intel® Dialogic® Digital Network Interface boards and the Network Termination Unit (NTU) are described in this Appendix.

Ordering Service

When 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.

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. Intel® 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	Recommendations/Requirements
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 NTU-to-board 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 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.

Appendix B:
ISDN Call Scenarios

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 scenarios are presented in the order listed below:

- BRI Channel Initialization and Start Up (User Side)
- BRI Channel Initialization and Start Up (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)
- 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 Rejects Outgoing Call
- Application Rejects Incoming Call (Synchronous Mode)
- Application Rejects Incoming Call (Asynchronous Mode)
- Glare (Call Collision)
- Simultaneous Disconnect (Any State)
- Network Facility Request - Vari-A-Bill (Asynchronous Mode)
- Network Facility Request - ANI-on-Demand (Incoming Call)
- Network Facility Request - Advice-of-Charge (Inbound & Outbound Calls)
- Application Disconnects Call (Synchronous Mode)

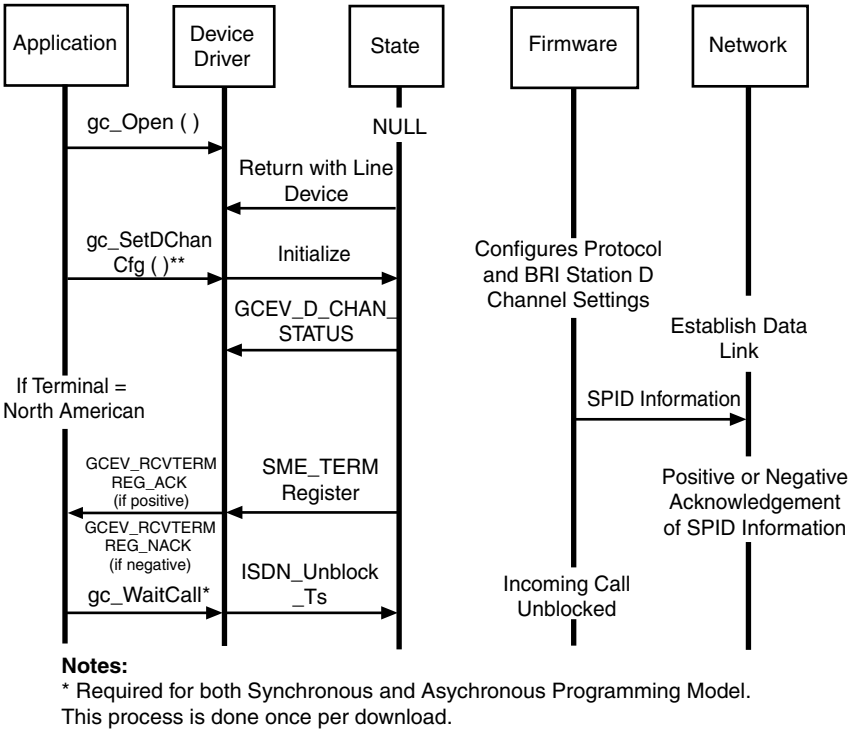
- Network Facility Request - Two B Channel Transfer (TBCT) (Synchronous Mode)
- Non-Call Associated Signaling (NCAS) (synchronous Mode)

Each scenario is presented in tabular format. The tables provide the following information:

- **Application** - Lists the functions called by the application
- **Device Driver** - Describes device driver activities including messages sent to firmware and events sent to application (internal activity)
- **State** - Lists current state (internal activity)
- **Firmware** - Describes firmware or firmware interface activities and messages sent to device driver and to network interface (internal activity).
- **Network** - Lists the D channel messages exchanged between the network and the device.

NOTE: The initials "PI" are used in the tables to indicate Progress Information.

BRI Channel Initialization and Start Up (User Side)



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

Figure 3. BRI Channel Initialization and Start Up (User Side)

BRI Channel Initialization and Start Up (Network Side)

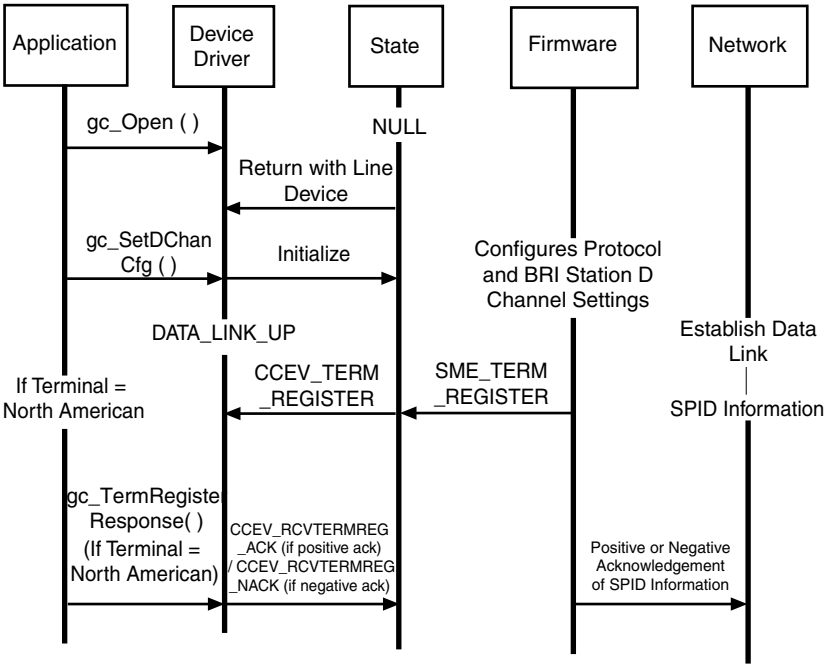
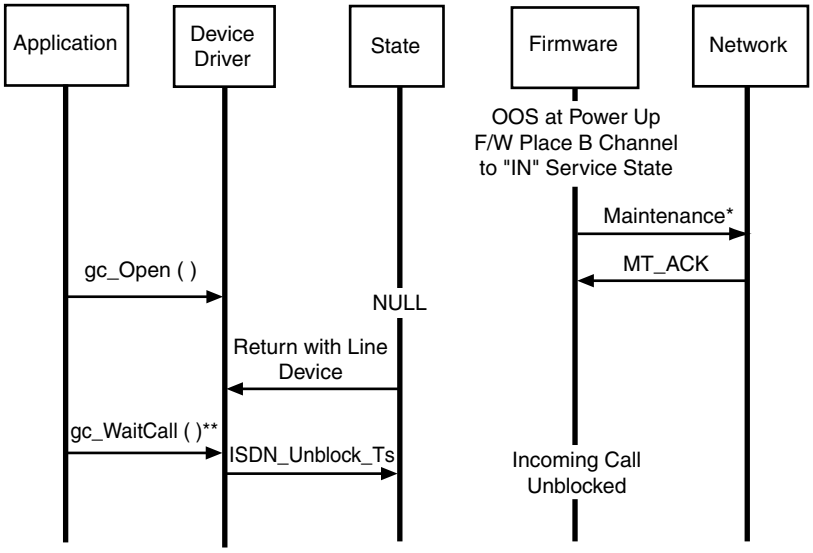


Figure 4. BRI Channel Initialization and Start Up (Network Side)

PRI Channel Initialization and Startup



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

Figure 5. PRI Channel Initialization and Startup

Network Initiated Inbound Call (Synchronous Mode)

The **gc_WaitCall( )** function must be issued for the next incoming call after the last call is terminated.

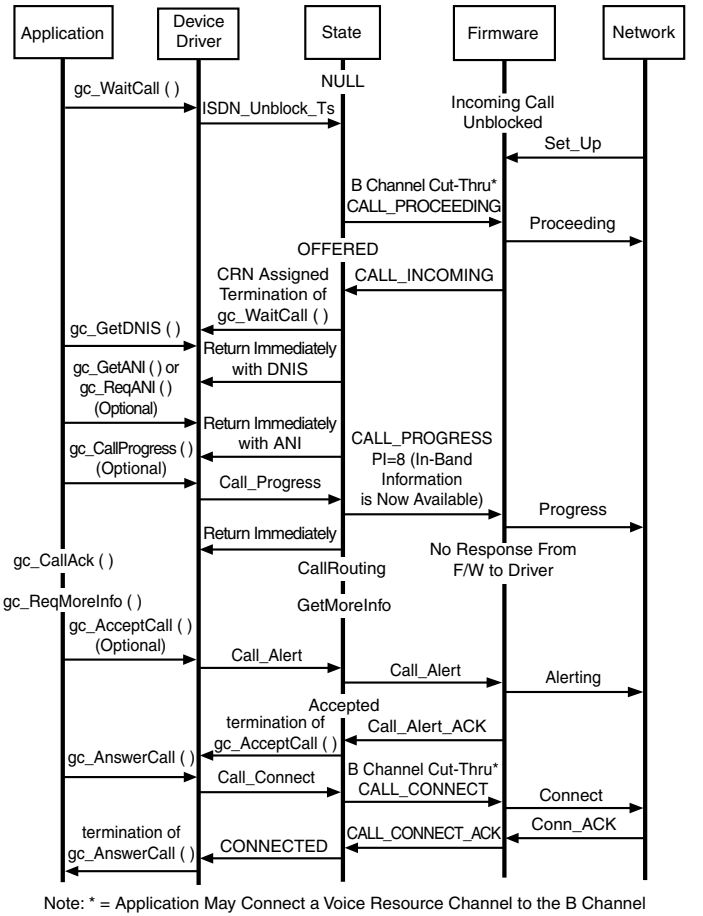


Figure 6. Network Initiated Inbound Call (Synchronous Mode)

Network Initiated Inbound Call (Asynchronous Mode)

Incoming call notification is received as an event. The **gc_WaitCall**( ) function should be issued once.

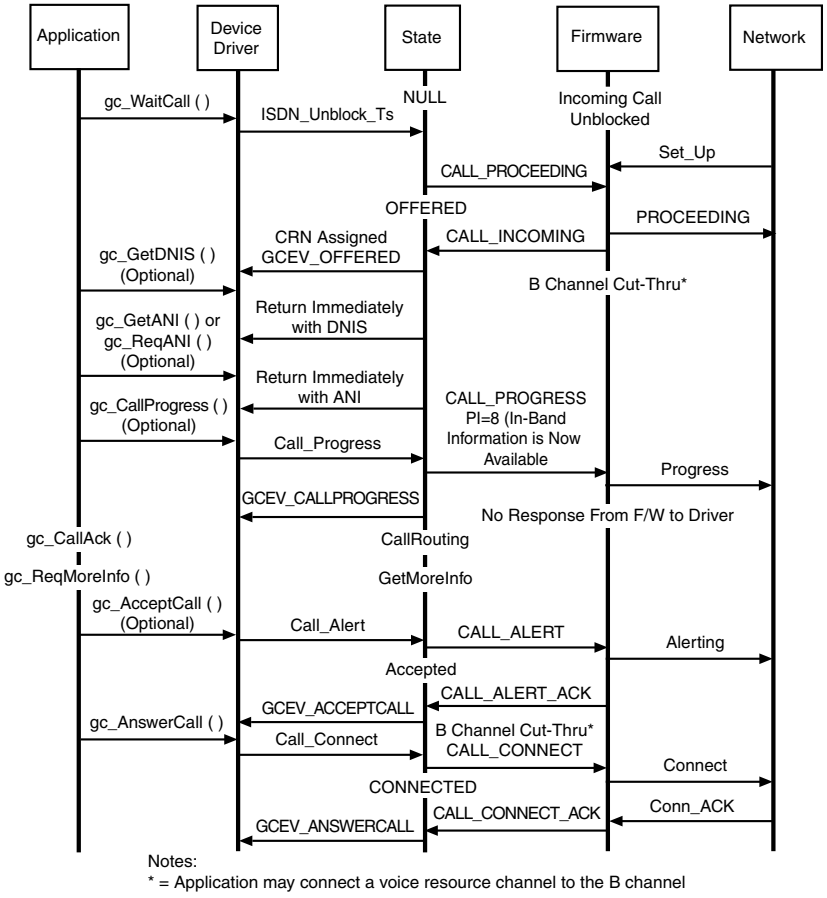
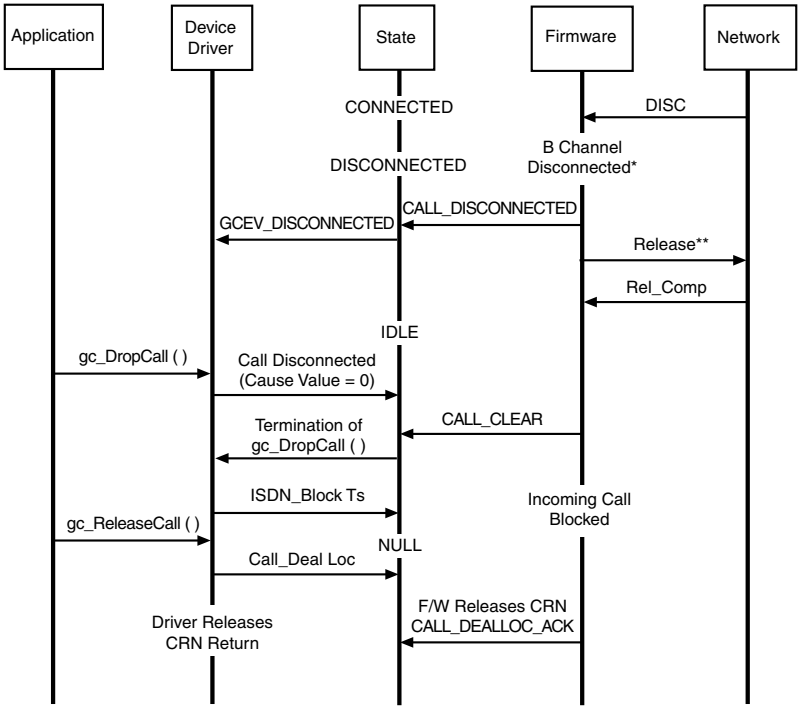


Figure 7. Network Initiated Inbound Call (Asynchronous Mode)

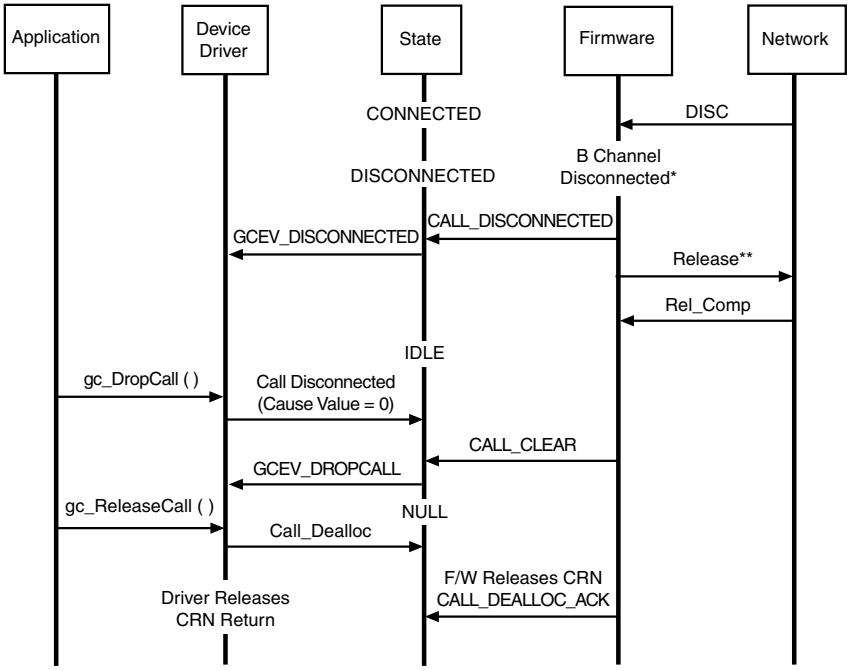
Network Terminated Call (Synchronous Mode)



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

Figure 8. Network Terminated Call (Synchronous Mode)

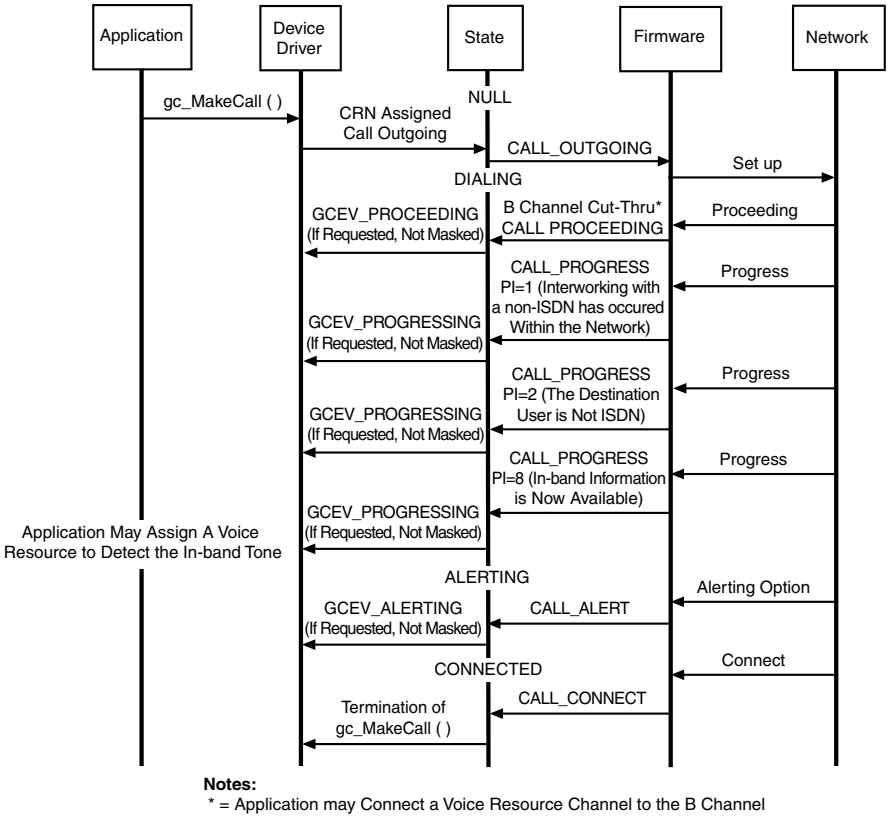
Network Terminated Call (Asynchronous Mode)



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

Figure 9. Network Terminated Call (Asynchronous Mode)

Application Initiated Outbound Call (Synchronous Mode)



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

Figure 10. Application Initiated Outbound Call (Synchronous Mode)

Application Initiated Outbound Call (Asynchronous Mode)

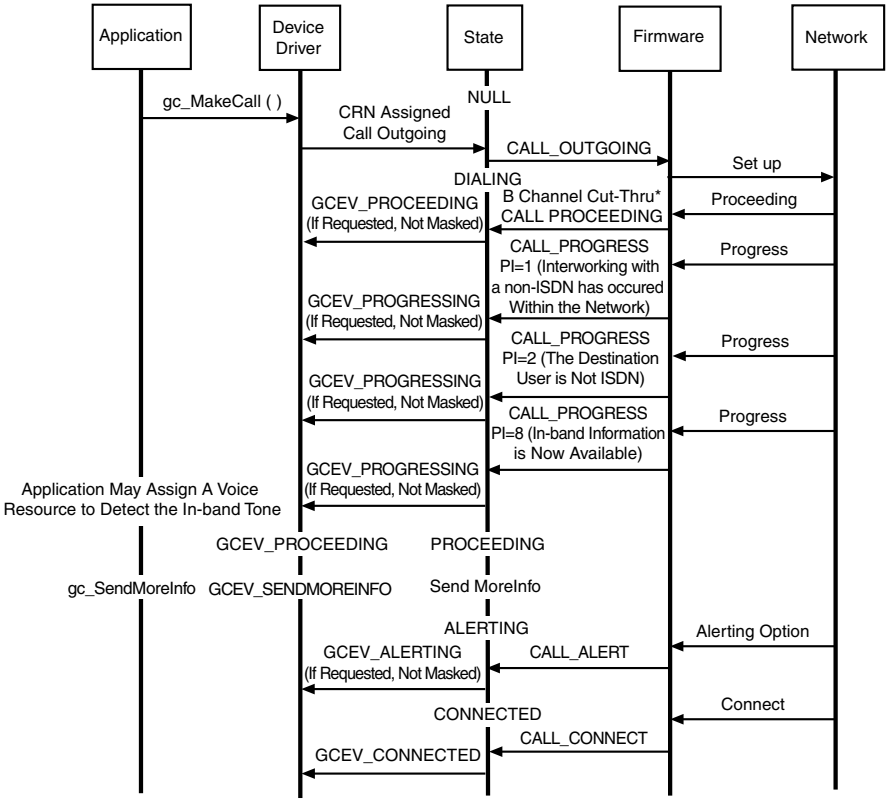


Figure 11. Application Initiated Outbound Call (Asynchronous Mode)

Aborting an Application-Initiated Call

NOTE: B channel negotiation is not currently available.

When a 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( )** attempt.

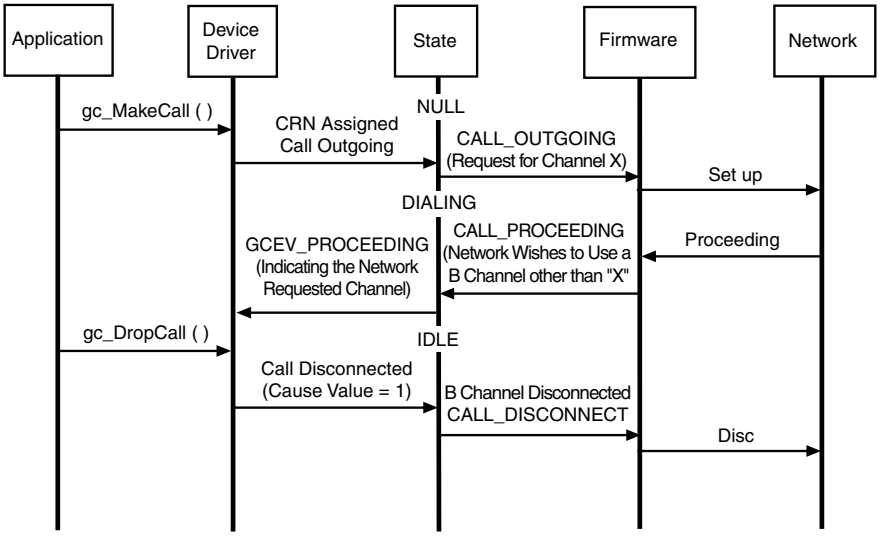


Figure 12. Aborting an Application-Initiated Call

Application Terminated Call (Synchronous Mode)

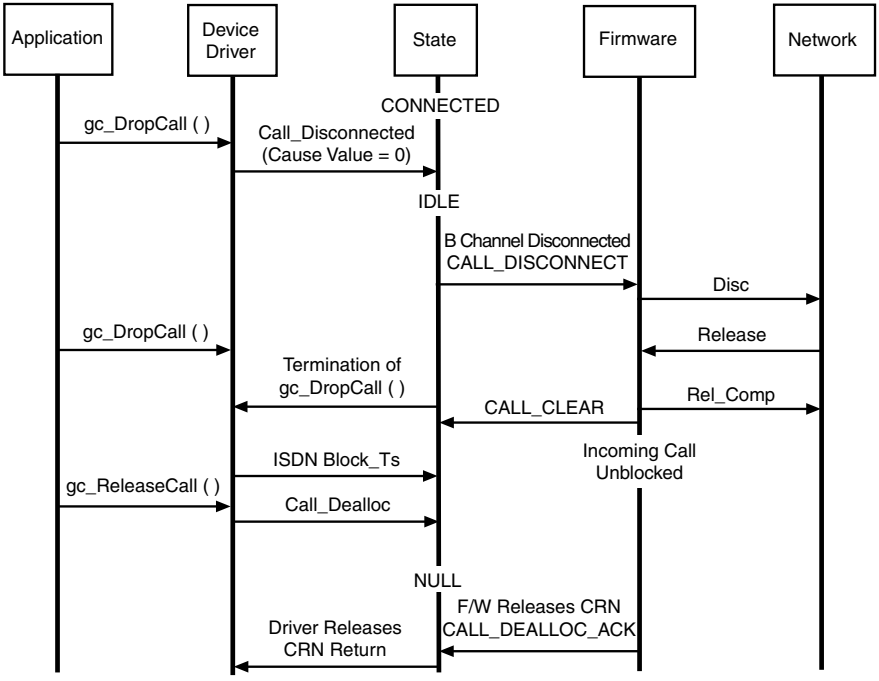


Figure 13. Application Terminated Call (Synchronous Mode)

Application Terminated Call (Asynchronous Mode)

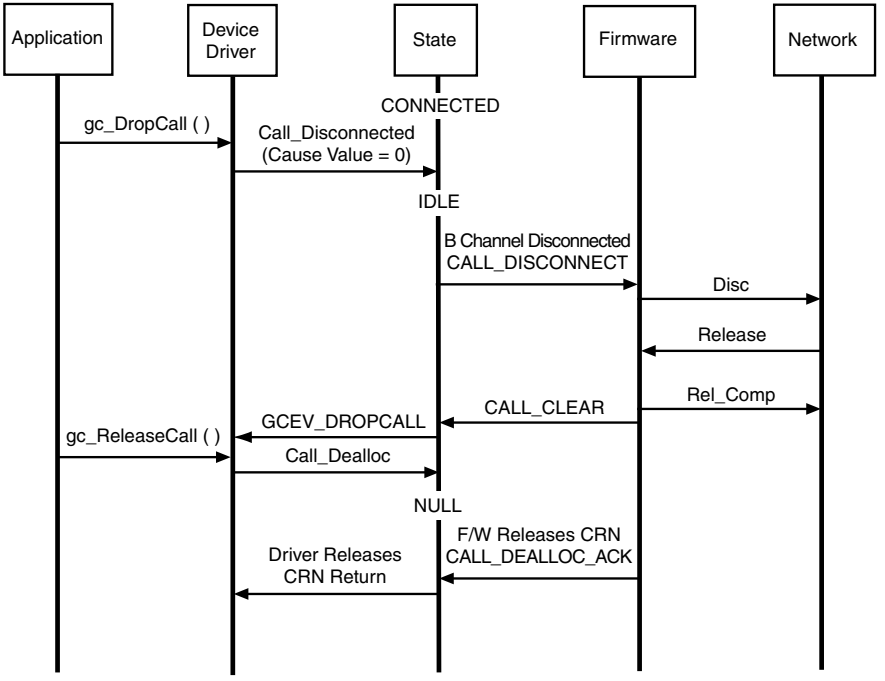


Figure 14. Application Terminated Call (Asynchronous Mode)

Network Rejects Outgoing Call

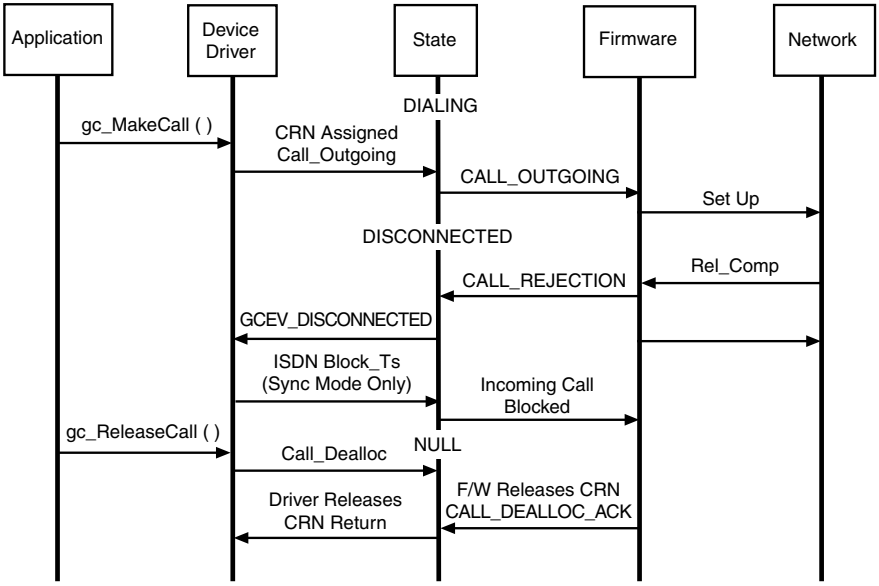


Figure 15. Network Rejects Outgoing Call

Application Rejects Incoming Call (Synchronous Mode)

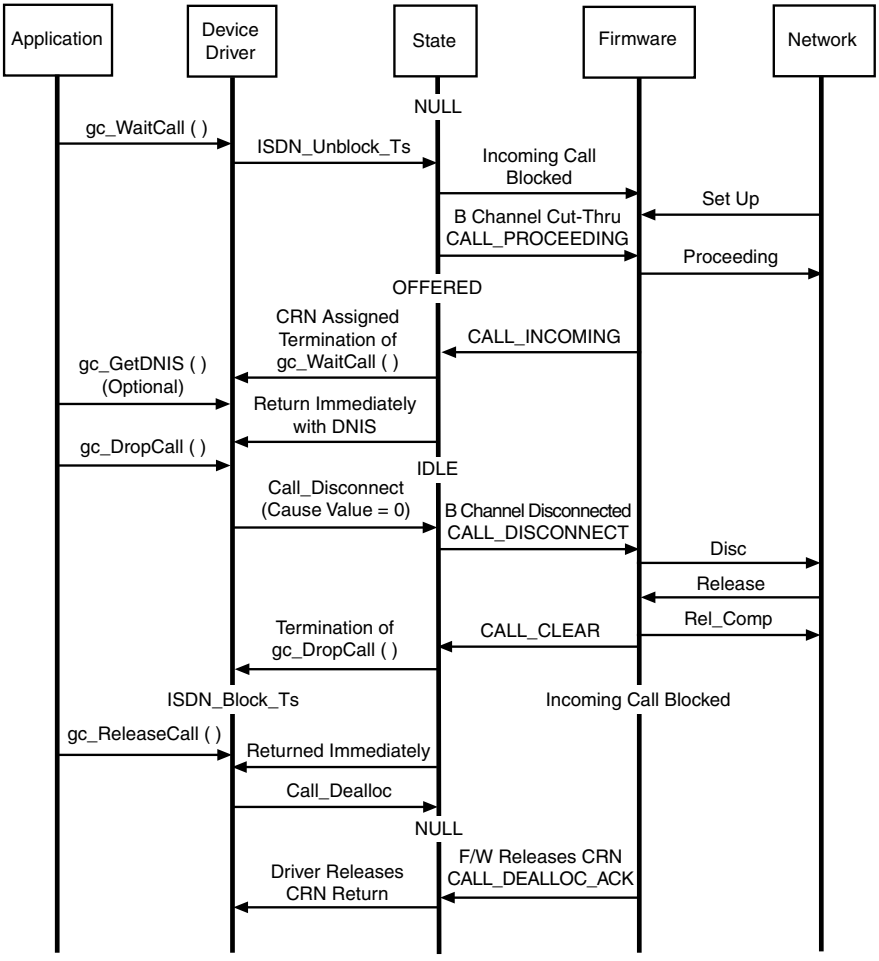


Figure 16. Application Rejects Incoming Call (Synchronous Mode)

Application Rejects Incoming Call (Asynchronous Mode)

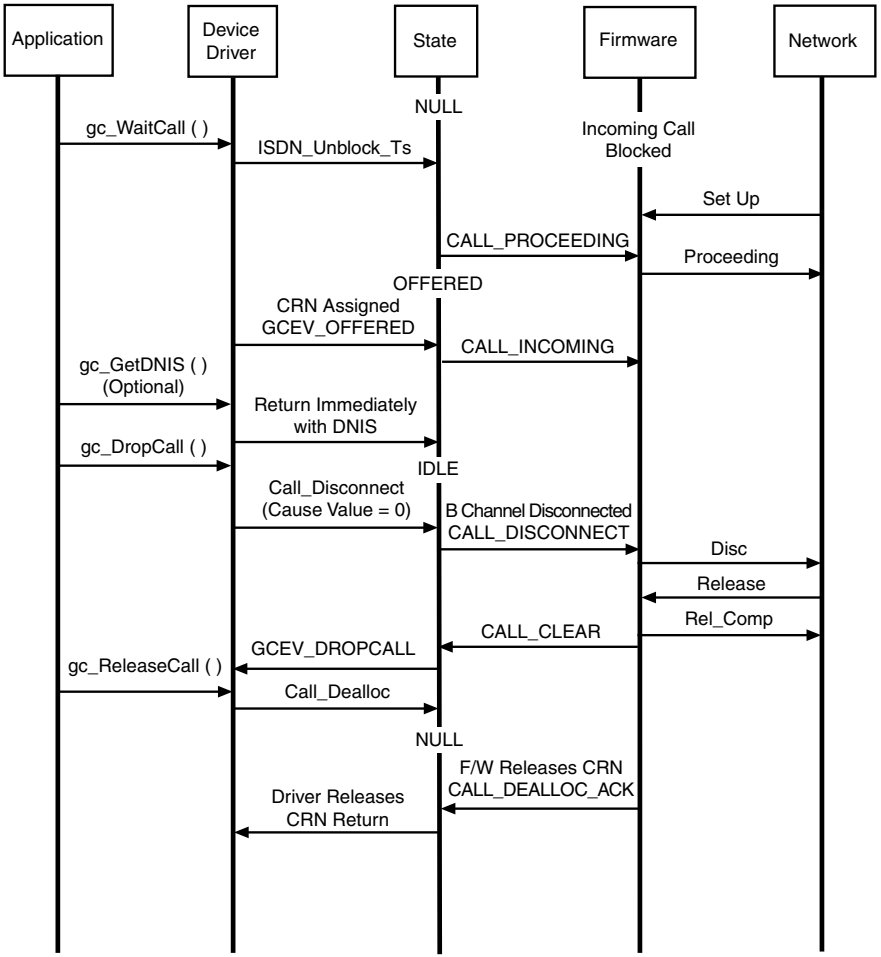


Figure 17. Application Rejects Incoming Call (Asynchronous Mode)

Glare (Call Collision)

The glare condition occurs when both an incoming and outgoing call requests the same time slot. When glare occurs, the incoming call is assigned the time slot. The following scenario illustrates handling a glare condition in asynchronous mode.

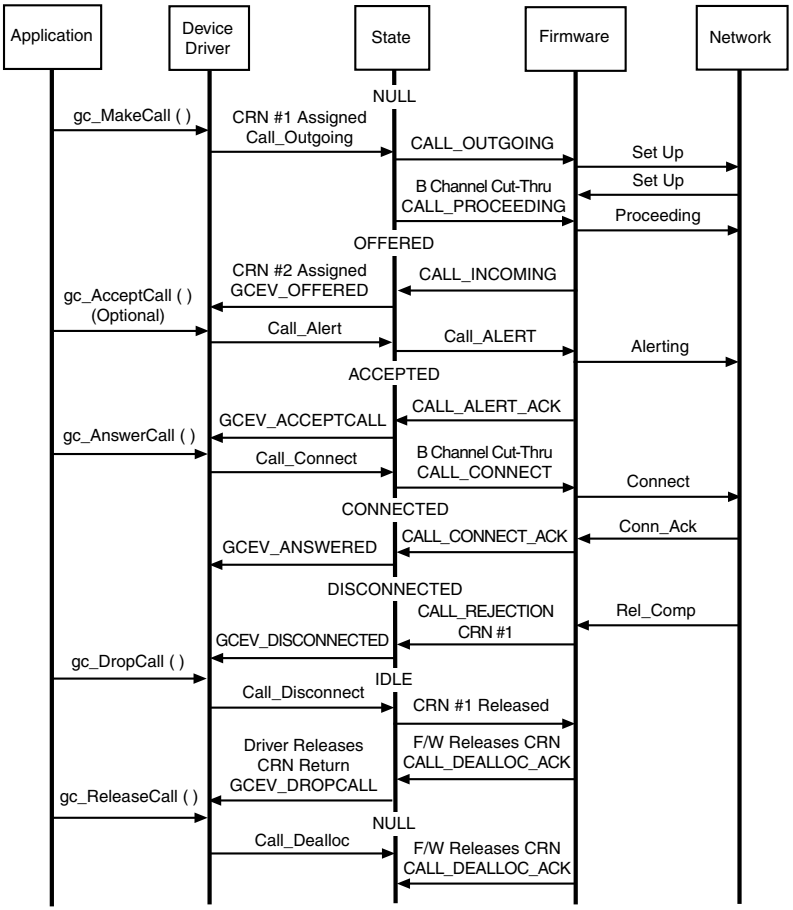


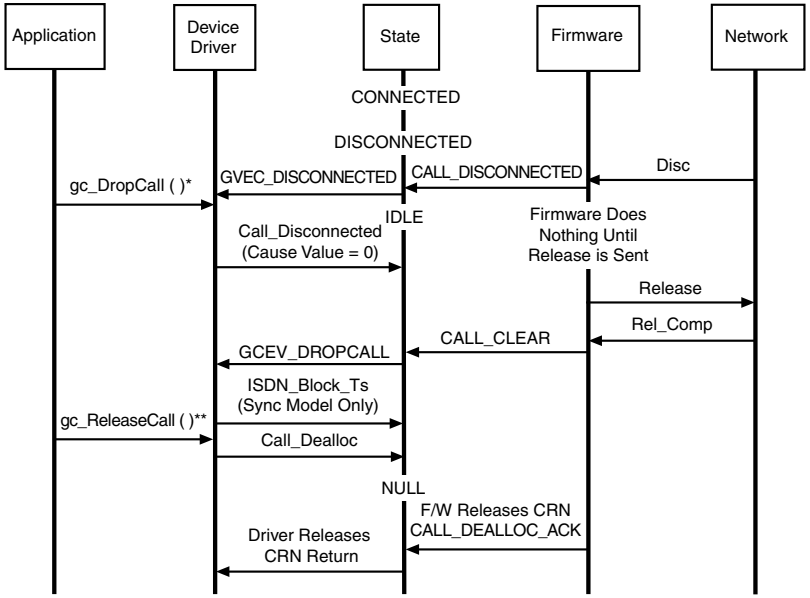
Figure 18. Glare (Call Collision)

Simultaneous Disconnect (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:

- 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.



Notes:
* = Application Should Set a "Drop Call" Flag
** = Application should ignore GCEV _DISCONNECTED if "Drop Call" Flag is Set
***= gc_ReleaseCall ( ) always clears "Drop Call" Flag

Figure 19. Simultaneous Disconnect (Any State) Scenario 1

Scenario 2:

- 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.

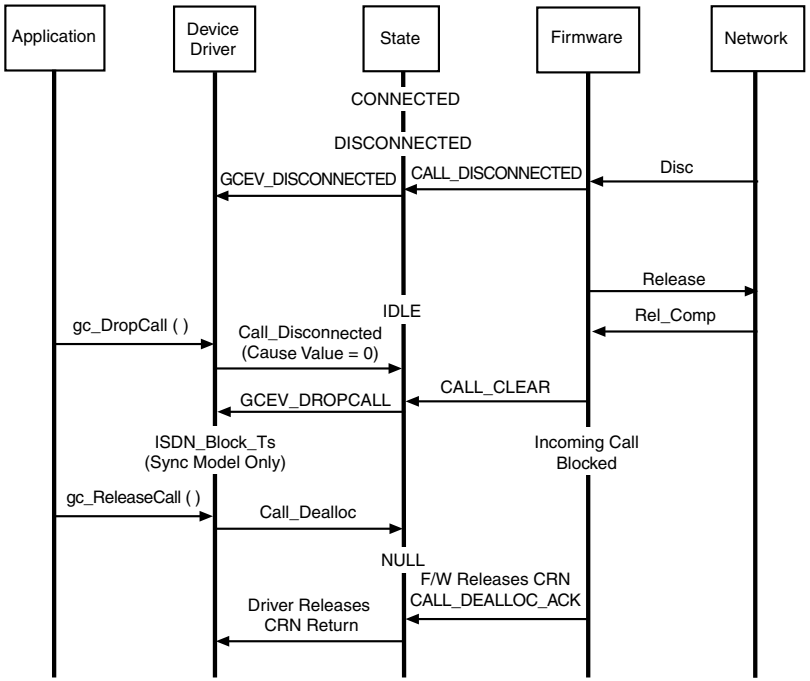
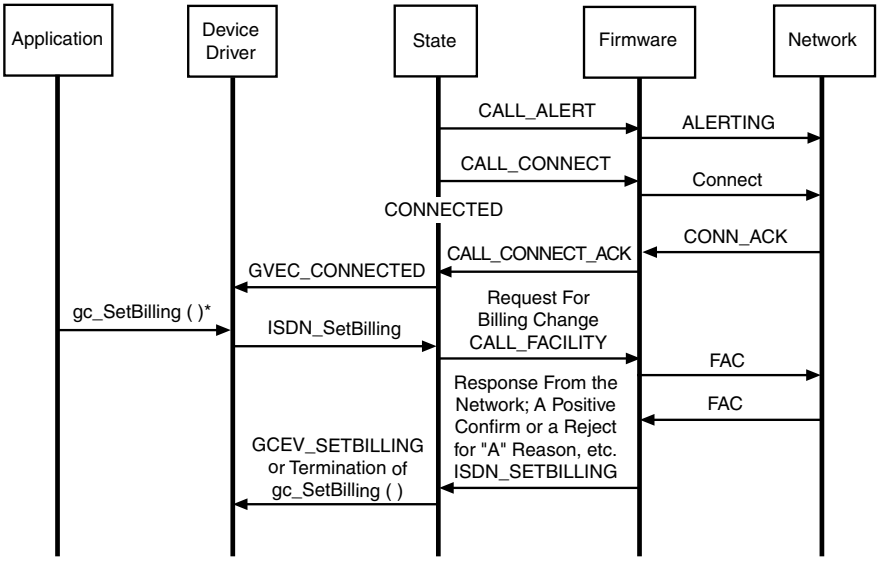


Figure 20. Simultaneous Disconnect (Any State) Scenario 2

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



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

Figure 21. Network Facility Request - Vari-A-Bill (Asynchronous Mode)

NOTE: Vari-A-Bill is a service option provided by AT&T.

Network Facility Request - ANI-on-Demand (Incoming Call)

The following scenario 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.

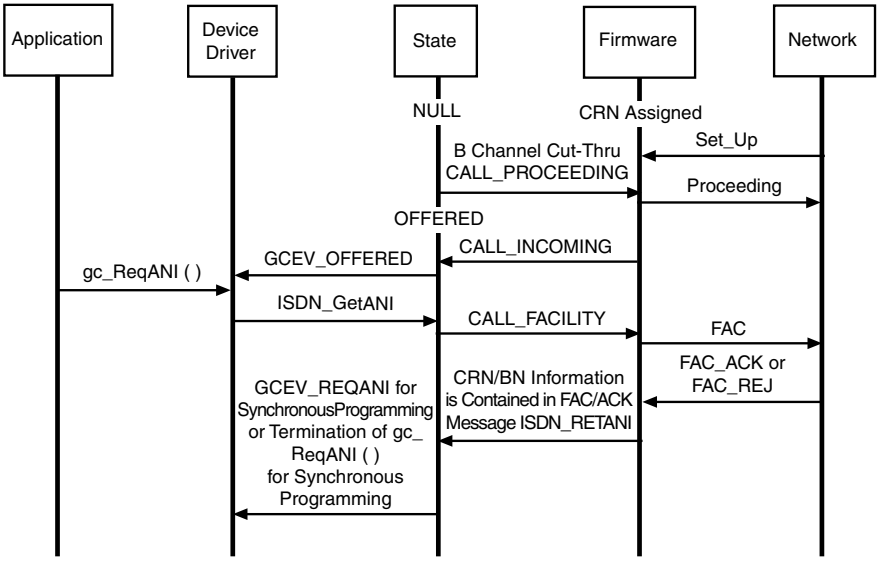


Figure 22. Network Facility Request - ANI-on-Demand (Incoming Call)

Network Facility Request - Advice-of-Charge (Inbound & Outbound Calls)

Advice-of-Charge is a service provided by AT&T.

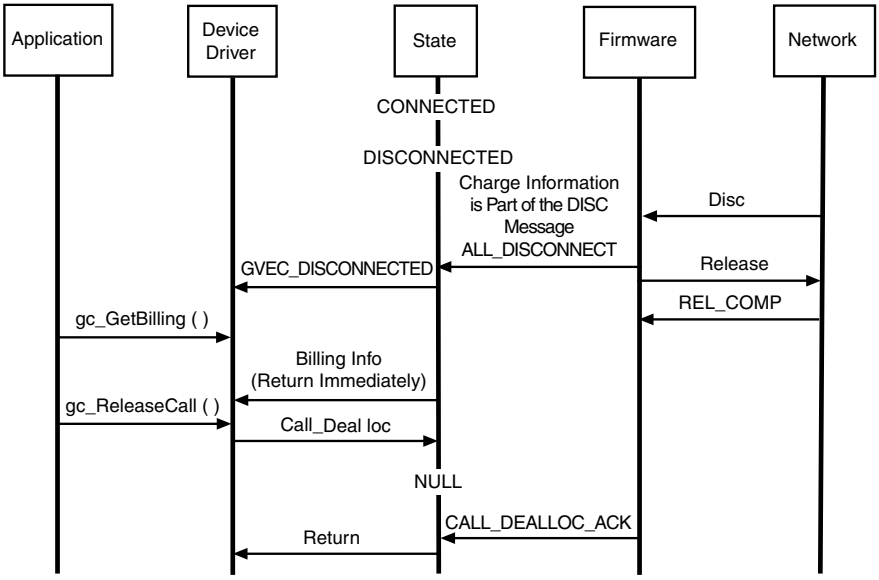


Figure 23. Network Disconnects Call (Asynchronous Mode)

Application Disconnects Call (Synchronous Mode)

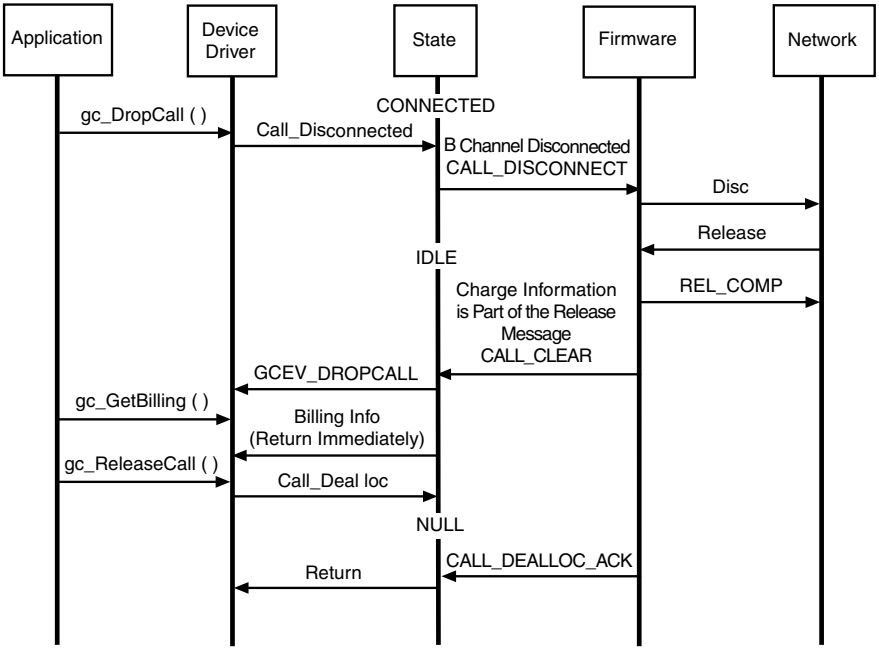


Figure 24. Application Disconnects Call (Synchronous Mode)

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

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 the 5ESS and 4ESS 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 Intel® Dialogic® firmware directly for the Network Call Reference Value. (See the *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 25 and Figure 26 provide line diagrams that illustrate the operation of the TBCT feature.

The sample code that follows Figure 29 uses the **gc_GetNetCRV( )** function to acquire the Call Reference Values (CRVs) of the two calls to be transferred.

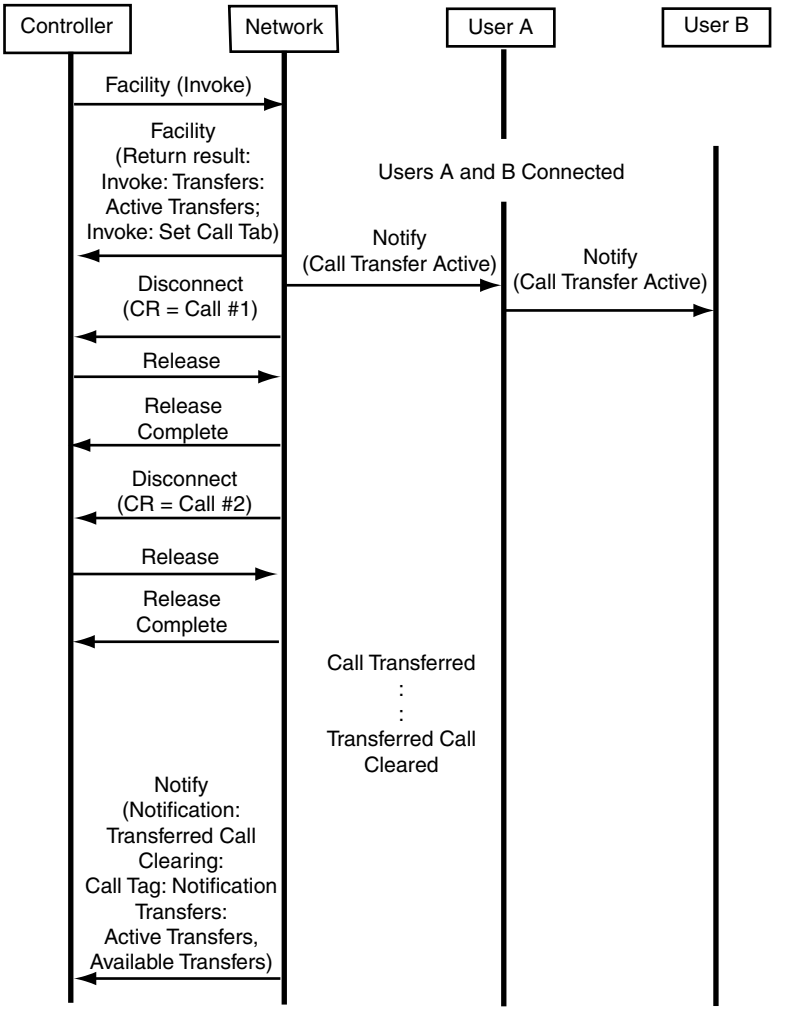


Figure 25. TBCT Invocation with Notification (Both Calls Answered)

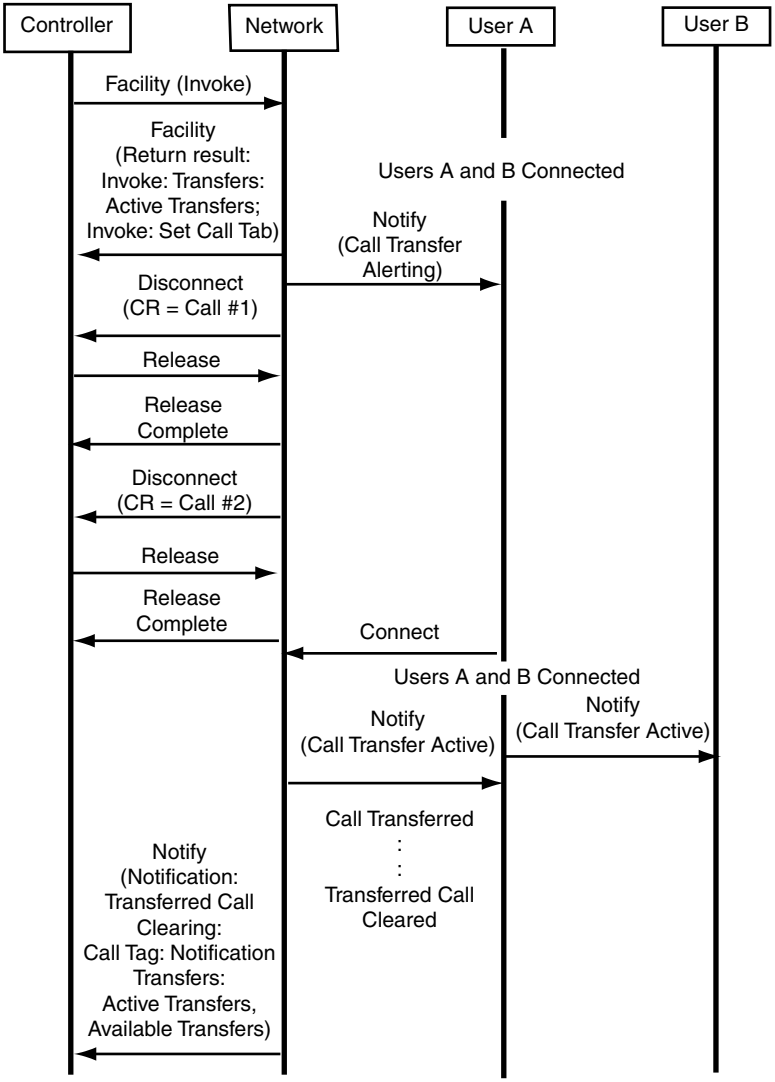


Figure 26. TBCT Invocation with Notification (Call 1 Answered/Call 2 Alerting)

Figure 27, Figure 28, and Figure 29 illustrate the procedures for initiating a TBCT. The scenario is followed by code samples that demonstrate the use of Intel® Dialogic® API in initiating a TBCT.

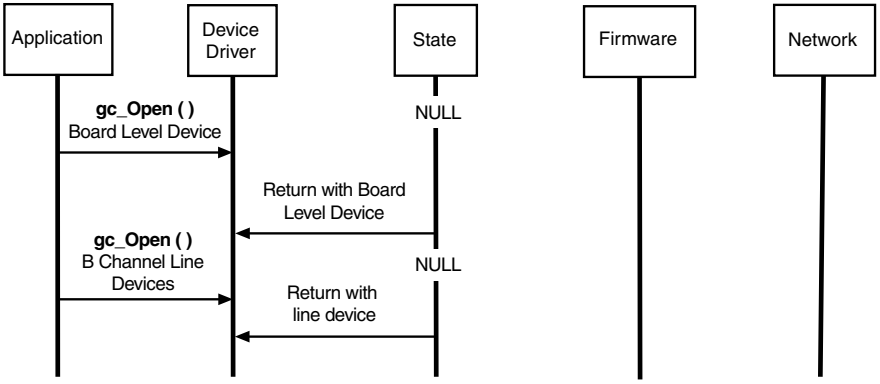


Figure 27. Synchronous Programming: Initiating TBCT

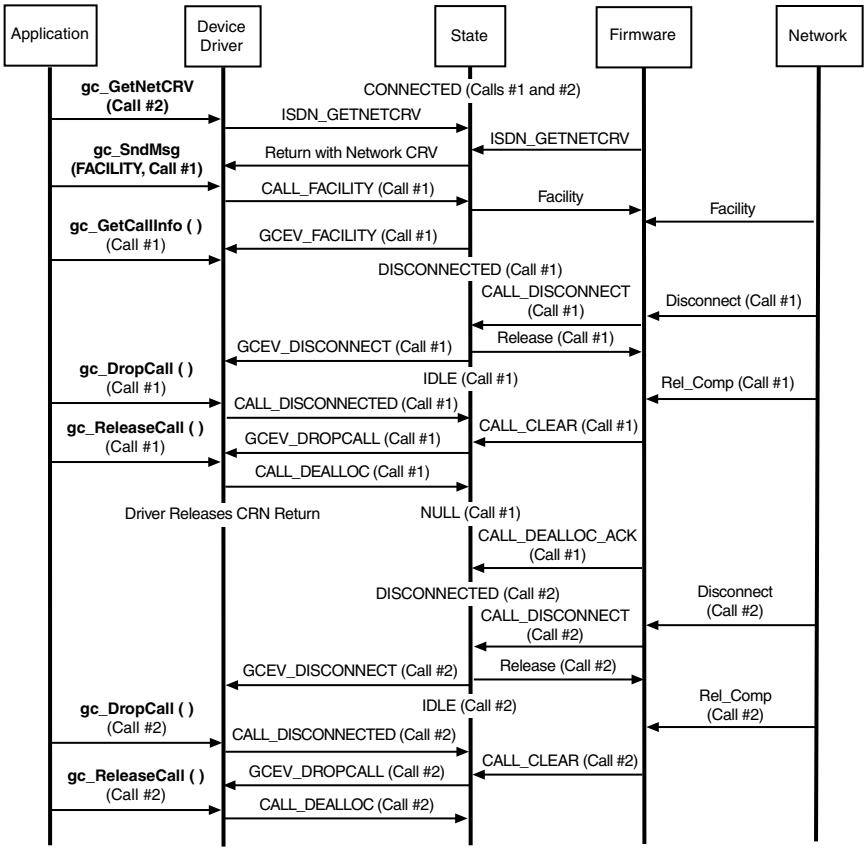


Figure 28. Synchronous Programming: Initiating TBCT (Users A and B Connected)

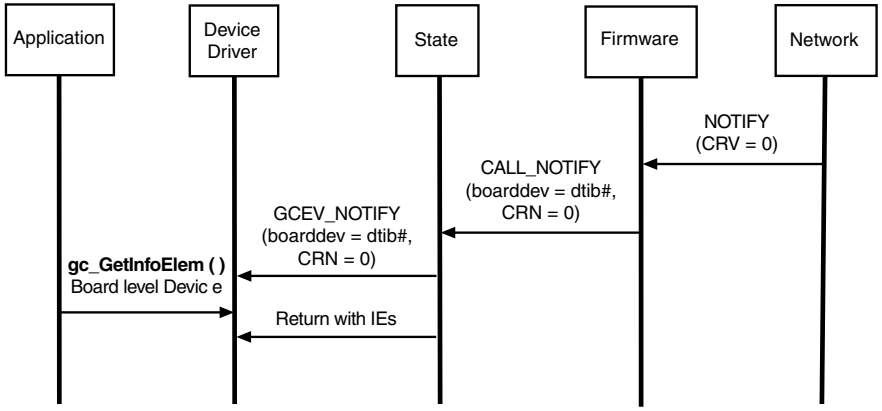


Figure 29. Synchronous Programming: Initiating TBCT (Users A and B Disconnected)

The following example code illustrates the use of the Global Call API at various stages of the TBCT call scenario.

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);
/*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);
/*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;
                 .
                 .
}
```

Non-Call Associated Signaling (NCAS) (synchronous Mode)

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. This feature is supported for the 5ESS protocol only.

Since NCAS calls are not associated with any B channel, applications should receive and transmit NCAS calls on the D channel line device. Once the NCAS connection is established, the application can transmit user-to-user messages using the CRN associated with the NCAS call. The Intel® Dialogic® software and firmware support 16 simultaneous NCAS calls per D channel.

Figure 30, Figure 31 and Figure 32 provide line diagrams that illustrate the operation of the NCAS feature.

The NCAS scenarios are shown in Figure 34, Figure 35, and Figure 36.

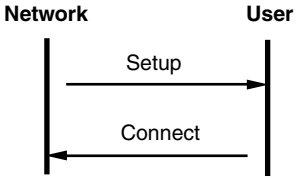


Figure 30. User-Accepted Network-Initiated NCAS Request

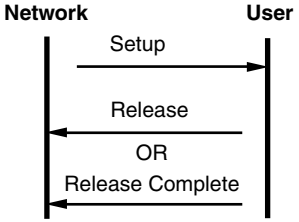


Figure 31. User-Rejected Network-Initiated NCAS Request

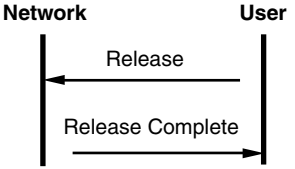


Figure 32. User-Disconnected NCAS Call

User-initiated call - in the following scenario, the user initiates and disconnects the NCAS call for dtiB1.

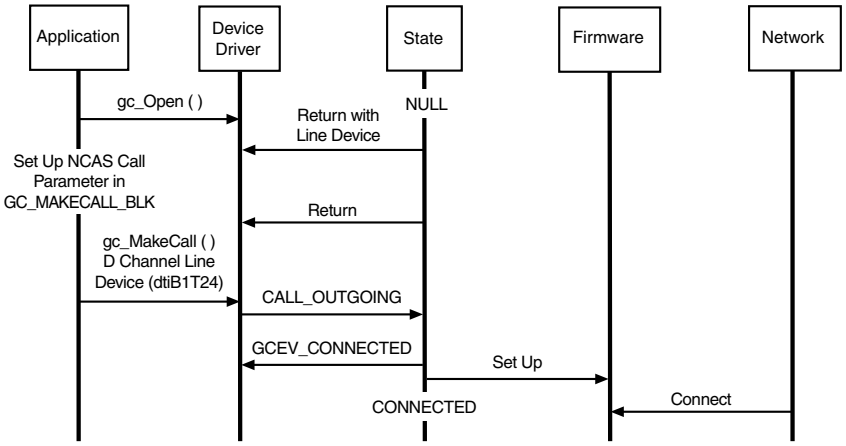


Figure 33. User-Initiated Call

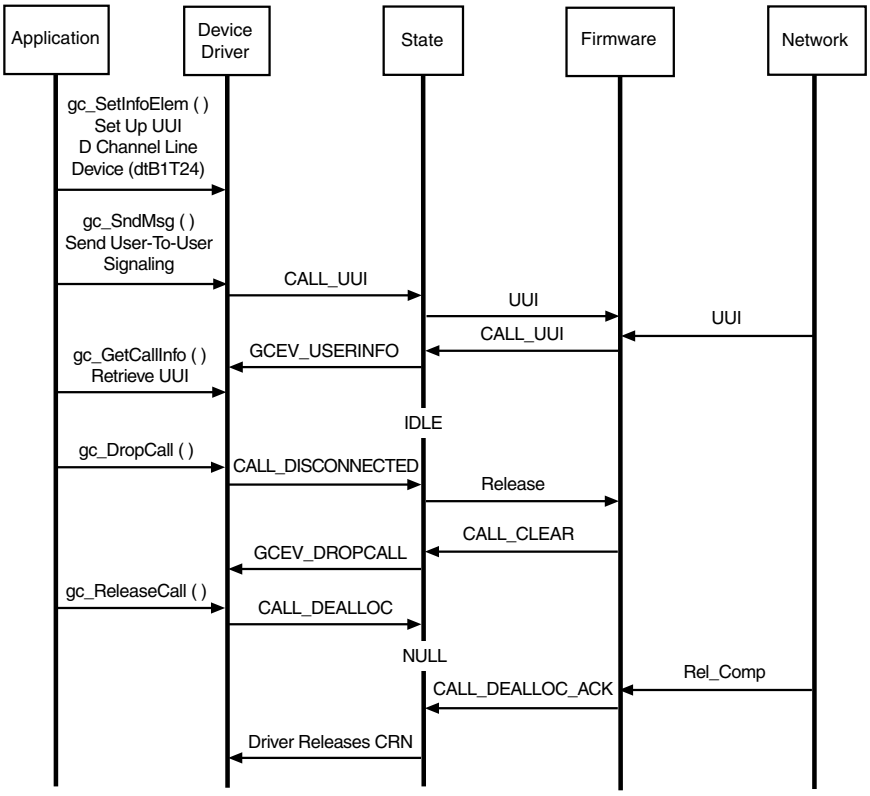


Figure 34. NCAS Call Connected

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

Opening a D channel line level device:

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

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_PACKET;
      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;
            .
            .
```

Network initiated call - in the following scenario, the network initiates and disconnects the NCAS call for dtiB1.

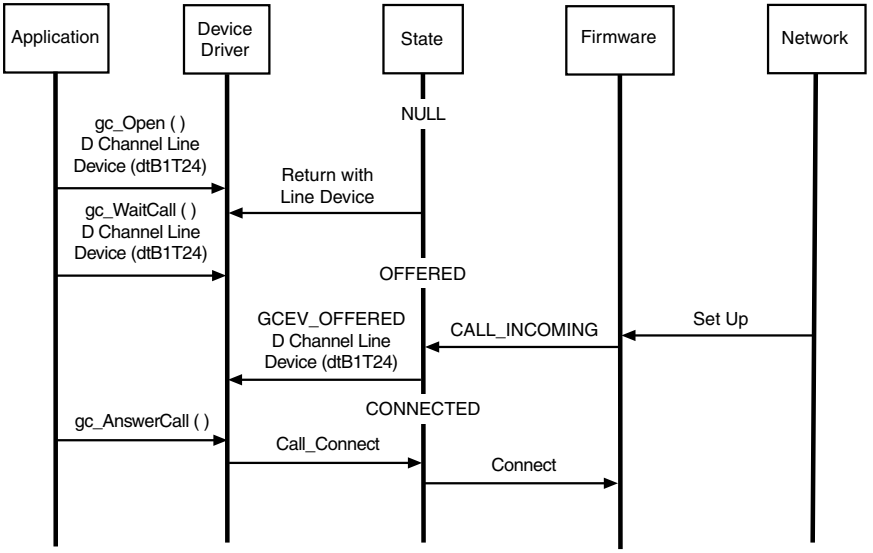


Figure 35. Network Initiated Call

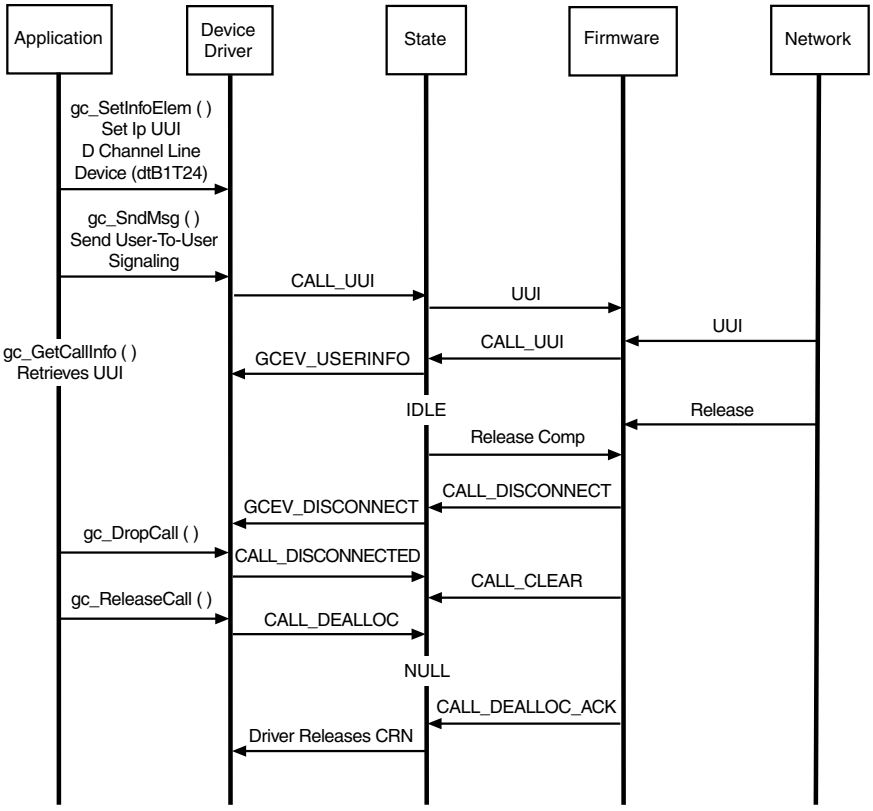


Figure 36. NCAS Call Connected

The following example code illustrates the use of the 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);
      .
```

Appendix C:
BRI Supplemental Services

The Global Call API functions allow BRI boards to perform the following Supplemental Services:

- 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 *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 1.2.1, “Framing”, on page 4 for more information). In order for the PBX to interpret the Information Elements as Supplemental Service requests, the Information Elements must be sent as Facility Messages.

The following functions can be used to send Facility Messages:

- **gc_Extension( )** with **ext_id** as CC_EXID_SndMsg - Sends a call state associated message to the PBX.

- **gc_Extension( )** with **ext_id** as 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 **ext_id** as CC_EXID_SndMsg and **CC_EXID_SndNonCallMsg( )** functions are 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( )** with **ext_id** as CC_EXID_cc_SndMsg or **gc_Extension( )** with **ext_id** as CC_EXID_SndNonCallMsg function with 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 **ext_id** as CC_EXID_SndMsg function with 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:

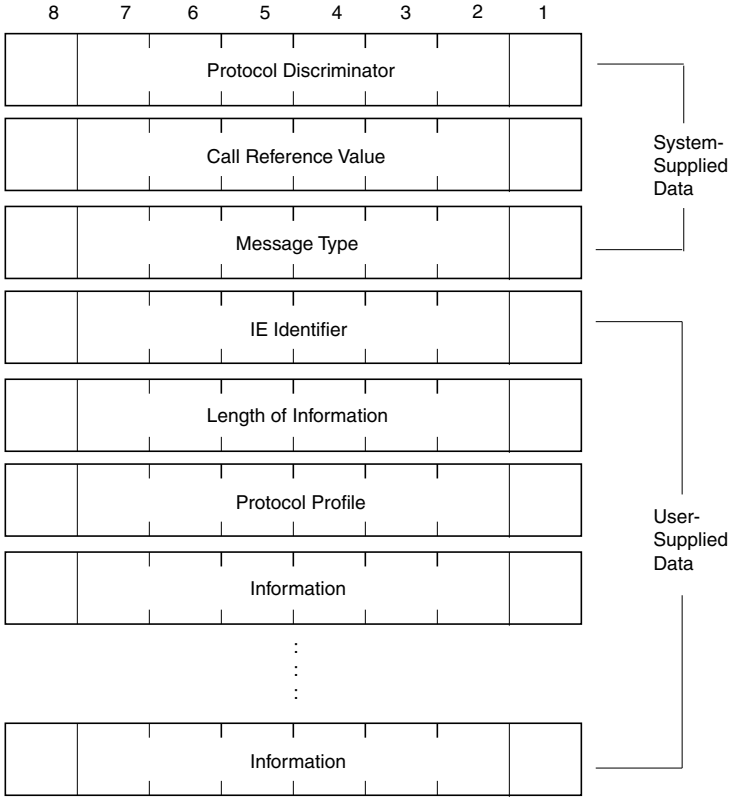


Figure 37. Information Element Format

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:

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							

Figure 38. Notify Message

Coding requirements for other supported Supplemental Services are listed in Table 58.

Table 58. 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. "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
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

Table 58. ETSI Specification Cross-Reference for Supplemental Services (Continued)

Supplementary Service/Description	ETS 300 Specification
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

Appendix D:
DPNSS IEs and Message Types

This appendix lists the information elements (IEs) and ISDN message types in the ISDN software library that support the DPNSS protocol. The scenarios are presented in the order listed below:

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 59. 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 60. 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 "diverted from" party. 2. A DIVERSION_ IE value in a GCEV_PROCEEDING event provides information about "divert to" party.
2. Data	Diversion IE Length	2 + length of Diversion Number	Number of data bytes in this IE

Table 60. Diversion IE (Continued)

Field	Description	Field Selection	Definition
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 61. 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 62. 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 63. 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.
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 64. 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 65. 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 66. 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 67. 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)

Table 67. Intrusion IE (Continued)

Field	Description	Field Selection	Definition
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 68. 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 69. Diversion Bypass IE

Field	Description	Field Selection	Definition
1. Length	Total bytes of the following data field	1	Required value

Table 69. Diversion Bypass IE

Field	Description	Field Selection	Definition
2. Data	Diversion Bypass IE ID	DIVERSION_BYPASS_IE	Use with the **gc_MakeCall( )** function to indicate that diversion is not allowed.

Table 70. 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 71. 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 72. 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 73. 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 74. 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
NOTE: NSI IE fields 5 and 6 can be repeated multiple times, as needed.			

Table 74. Network Specific Indications (NSI) IE

Field	Description	Field Selection	Definition
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 75. 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 76. 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
3. Data	Intrude IE Length	1	Number of data bytes for this IE
4. Data	Intrude Type	INTRUDE INTRUDE_WITHDRAW	INTRUDE INTRUDE_WITHDRAW

Table 77. 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 78. 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 79. 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
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

Appendix E:
DPNSS Call Scenarios

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

- A table that illustrates the Global Call 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 - Normal
- Executive Intrusion - With Prior Validation
- Hold and Retrieve - Locally Initiated
- Hold and Retrieve - Remotely Initiated
- Local Diversion - Outbound
- Local Diversion - Inbound
- Remote Diversion - Outbound
- Remote Diversion - Inbound
- Transfer
- Virtual Call - Outbound
- Virtual Call - Inbound

Executive Intrusion - Normal

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 67, "Intrusion IE", on page 278 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.

Executive Intrusion - With Prior Validation

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 67, “Intrusion IE”, on page 278 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 for more information. See Table 59, “Intrusion IE”, on page 275 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 *Global Call API Library Reference* and Section 2.36, “gc_SndMsg( )”, on page 146 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.

Hold and Retrieve - Locally Initiated

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 unroute 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.

Hold and Retrieve - Remotely Initiated

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.

Local Diversion - Outbound

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 *Global Call API Library Reference* and Section 2.36, "gc_SndMsg( )", on page 146 for ISDN-specific information.
3. The application answers the call using the **gc_Answer( )** function.
4. A call answered (GCEV_ANSWERED) event is received.

Local Diversion - Inbound

Step	API	Action/Result	Event
1	**gc_MakeCall( )**	--->	

Step	API	Action/Result	Event
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 Section 60, “Diversion IE”, on page 275 for the Diversion IE format.
3. A call connected (GCEV_CONNECTED) event is received and the call is established.

Remote Diversion - Outbound

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 succeeded	---

Step	API	Action/Result	Event
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 60, "Diversion IE", on page 275 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 *Global Call API Library Reference* and Section 2.33, "gc_SetUserInfo( )", on page 144 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_DISCONNECT) event from Party 3. The call was not diverted.

Remote Diversion - Inbound

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 75, "SndMsg_Divert", on page 282 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.

Transfer

Step	API	Action/Result	Event
1		<---	GCEV_OFFERED (CRN 1)
2	**gc_AnswerCall( )** (CRN 1)	--->	

Step	API	Action/Result	Event
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)

Step	API	Action/Result	Event
16	**gc_SndMsg( )** (SndMsg_Transit, CRN 1)	--->	
17		<---	GCEV_DISCONNECT (CRN 1)
18	**gc_DropCall( )** (CRN 1)	--->	
19		<---	GCEV_DROPCALL (CRN 1)
20	**gc_ReleaseCall( )** (CRN 1)	--->	
21		<---	GCEV_DISCONNECT (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.
4. Some switches may not support holding a call.
5. Party 2 receives a call on hold acknowledge (GCEV_HOLDACK) event.
6. 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 70, "Inquiry IE", on page 280 for the Inquiry IE format.

7. Party 2 receives a call proceeding (GCEV_PROCEEDING) event with an Inquiry IE from Party 3. See Table 70, "Inquiry IE", on page 280 for the Inquiry IE format.
8. Party 2 receives a call connected (GCEV_CONNECTED) event with Inquiry IE from Party 3. See Table 70, "Inquiry IE", on page 280 for the Inquiry IE format.
9. 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 78, "SndMsg_Transfer", on page 284 for the message format.
10. 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 78, "SndMsg_Transfer", on page 284 for the message format.
11. Party 2 receives a transfer acknowledge (GCEV_TRANSFERACK) event from Party 1.
12. Party 2 receives a transfer acknowledge (GCEV_TRANSFERACK) event from Party 3. Transfer completed. At this time, Party 2 loses control of the call.
13. 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.
14. 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.
15. Party 2 sends content of the Transit IE (unchanged) from Party 1 to Party 3 using the **gc_SndMsg( )** function. See Table 79, "SndMsg_Transit", on page 284 for the message format.
16. 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.
17. Party 2 sends content of Transit IE (unchanged) from Party 3 to Party 1 using the **gc_SndMsg( )** function. See Table 79, "SndMsg_Transit", on page 284 for the message format.
18. Party 2 receives a disconnect all (GCEV_DISCONNECT) event from Party 1.
19. Party 2 drops the call to Party 1 using the **gc_DropCall( )** function.
20. Party 2 receives a drop call (GCEV_DROPCALL) event from Party 1.
21. Party 2 releases the call to Party 1 using the **gc_ReleaseCall( )** function.

22. Party 2 receives a disconnect call (GCEV_DISCONNECTED) event from Party 3.
23. Party 2 drops the call to Party 3 using the **gc_DropCall( )** function.
24. Party 2 receives a drop call (GCEV_DROPCALL) event from Party 3.
25. 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.

Virtual Call - Outbound

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 72, “Virtual Call IE”, on page 280 for the format of the Virtual Call IE.
2. The application receives a call disconnected (GCEV_DISCONNECT) 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.

Virtual Call - Inbound

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.

Index

Numerics

A

B

C

D

E

F

G

L

M

N

O

P

R

S

T

U

V

W