Global Call IP over Embedded Stack Technology User's Guide

for Linux and Windows

05-1940-001

COPYRIGHT NOTICE

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel® products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This document as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Dialogic, an Intel company. Dialogic, an Intel company assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without express written consent of Dialogic, an Intel company.

Some names, products, and services mentioned herein are the trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Other names and brands may be claimed as the property of others.

Publication Date: January 2003

Part Number: 05-1940-001

Intel Corporation
Telecommunication and Embedded Group
1515 Route 10
Parsippany, NJ 07054

For **Technical Support**, visit the Dialogic support website at:
http://support.dialogic.com

For **Sales Offices** and other contact information, visit the main Dialogic website at:
http://www.dialogic.com

Table of Contents

List of Figures

List of Tables

Preface

This guide is for users of the Global Call API writing applications that use embedded IP technology. The Global Call API provides call control capability and supports IP Media control capability. This guide provides Global Call IP-specific information only and should be used in conjunction with the *Global Call Application Developer's Guide* and the *Global Call API Software Reference* that describe the generic behavior of the Global Call API.

Organization of this Guide

This guide contains the following chapters:

Chapter 1 Provides a overview of VoIP technology and the H.323 standard for novice users.

Chapter 2 Describes how Global Call can be used with embedded IP technology and provides an overview of the architecture.

Chapter 3 Describes the additional functionality of specific Global Call API functions used for embedded IP technology.

Chapter 4 Describes how to use Global Call to perform IP-specific tasks, such as setting call related information, setting Quality of Service (QoS) thresholds, and others.

Chapter 5 Provides guidelines for building Global Call applications that use embedded IP technology.

Chapter 6 Provides a reference for IP-specific parameter set IDs and their associated parameter IDs.

Chapter 7 Provides a data structure reference for Global Call IP-specific data structures.

Chapter 8 Provides descriptions of IP-specific event cause codes.

Chapter 9 Gives the formats for called and calling party address lists.

A Glossary and an Index can be found at the end of the document.

Global Call Product Support

The Global Call software provides a consistent interface across Intel products interfaces to various networks (for example, T-1 ISDN, E-1 ISDN, E-1 CAS and T-1 robbed bit, SS7 and IP). See the *Release Guide* for your system release software for an up-to-date list of the products that support Global Call and provide IP signaling capabilities.

1. VoIP and H.323 Overview

1.1. VoIP Overview

Voice over IP (VoIP) can be described as the ability to make telephone calls and send faxes over IP-based data networks with a suitable Quality of Service (QoS). The voice information is sent in digital form using discrete packets rather than via dedicated connections as in the circuit-switched Public Switch Telephone Network (PSTN). At the time of writing this document, there are two major international groups defining standards for VoIP:

- International Telecommunications Union (ITU) - The ITU has defined the H.323 standard, which covers VoIP.
- Internet Engineering Task Force (IETF) - The IETF has defined drafts of the following RFC (Request for Comment) documents:
  - RFC 2543, the Session Initiation Protocol (SIP)
  - RFC 27053, the Media Gateway Control Protocol (MGCP)
  - RFC 3015, Megaco Protocol Version 1.0 (Megaco)

The H.323 standard was developed in the mid 1990s and is more mature than any of the protocols mentioned above.

1.2. H.323 Overview

The H.323 specification is an umbrella specification for the implementation of packet-based multimedia over IP networks that cannot guarantee Quality of Service (QoS).

1.2.1. H.323 Entities

The H.323 specification defines the entity types in an H.323 network including:

- **Terminal** - An endpoint on an IP network, that supports the real-time, two-way communication with another H.323 entity. A terminal supports voice coders/decoders (codecs) and setup and control signaling.

- **Gateway** - Provides the interface between a packet-based network (for example, an IP network) and a circuit-switched network (for example, the PSTN). A gateway translates communication procedures and formats between networks. It handles call setup and teardown and the compression and packetization of voice information.
- **Gatekeeper** - Manages a collection of H.323 entities in an H.323 zone controlling access to the network for H.323 terminals, Gateways and MCUs and providing address translation. A zone can span a wide geographical area and include multiple networks connected by routers and switches. Typically there is only one gatekeeper per zone, but there may be an alternate gatekeeper for backup and load balancing. Typically, endpoints, such as terminals, gateways and other gatekeepers register with the gatekeeper.
- **Multipoint Control Unit (MCU)** - An endpoint that support conferences between three or more endpoints. An MCU can be a stand-alone unit or integrated into a terminal, gateway, or gatekeeper. An MCU consists of:
  - **Multipoint Controller (MC)** - Handles control and signaling for conferencing support.
  - **Multipoint Processor (MP)** - Receives streams from endpoints, processes them, and returns them to the endpoints in the conference.

Figure 1 shows the entities in a typical H.323 network.

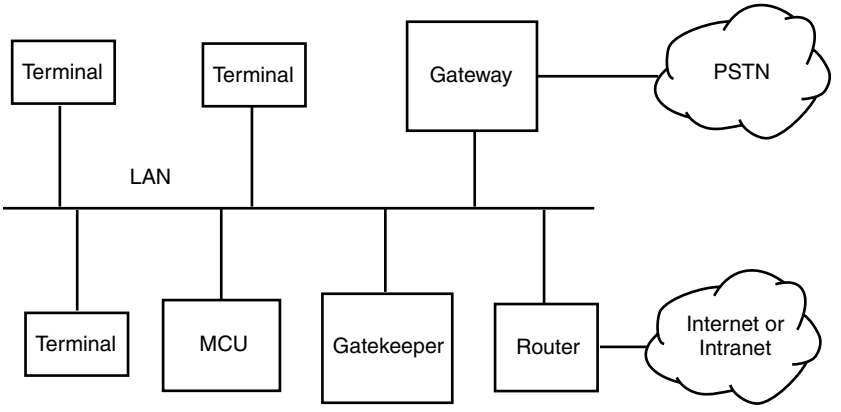


Figure 1. Typical H.323 Network

1.2.2. H.323 Protocol Stack

The H.323 specification is an umbrella specification for the many different protocols that comprise the overall H.323 protocol stack. Figure 2 shows the H.323 protocol stack.

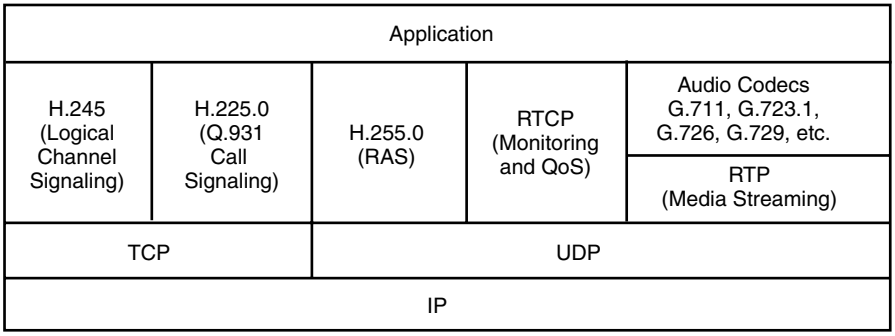


Figure 2. H.323 Protocol Stack

The purpose of each protocol is summarized briefly as follows:

- **H.245** - Specifies messages for opening and closing channels for media streams, and other commands, requests and indications.
- **Q.931** - Defines signaling for call setup and call teardown.
- **H.225.0** - Specifies messages for call control including signaling, Registration Admission and Status (RAS), and the packetization and synchronization of media streams.
- **Real Time Protocol (RTP)** - The RTP specification (RFC 18890) is an IETF draft standard that defines the end-to-end transport of real-time data. RTP does not guarantee quality of service (QoS) on the transmission. However, it does provides some techniques to aid the transmission of isochronous data including:
  - Information about the type of data being transmitted
  - Time stamps
  - Sequence numbers

- **Real Time Control Protocol (RTCP)** - RTCP is part of the RTP specification (RFC 18890) and defines the end-to-end monitoring of data delivery and QoS by providing information such as:
  - Jitter, that is, the variance in the delays introduced in transmitting data over a wire
  - Average packet loss

The H.245, Q.931 and H.225.0 combination provide the signaling for the establishment of a connection, the negotiation of the media format that will be transmitted over the connection, and call teardown at termination. As indicated in Figure 2, the call signaling part of the H.323 protocol is carried over TCP, since TCP guarantees the in-order delivery of packets to the application.

The RTP and RTCP combination is for media handling only. As indicated in Figure 2, the media part of the H.323 protocol is carried over UDP and therefore there is no guarantee that all packets will arrive at the destination and be placed in the correct order.

1.2.3. Codecs

RTP and RTCP data is the payload of a User Datagram Protocol (UDP) packet. Analog signals coming from the an endpoint are converted into the payload of UDP packets by codecs (coders/decoders). The codecs perform compression and decompression on the media streams.

Different types of codecs provide varying sound quality. The bit rate of most narrow-band codecs is in the range 1.2 Kbits/s to 64 Kbits/s. The higher the bit rate the better the sound quality. Some of the most popular codecs are:

- **G.711** - Provides a bit rate of 64 Kbits/s.
- **G.723.1** - Provides bit rates of either 5.3 or 6.4 Kbits/s. Voice communication using this codec typically exhibits some form of degradation.
- **G.729** - Provides a bit rate of 8 Kbits/s. This codec is very popular for voice over frame relay and for V.70 voice and data modems.

1.3. Basic H.323 Call Scenario

A simple H.323 call scenario can be described in five phases:

- Call Setup
- Capability Exchange
- Call Initiation
- Data Exchange
- Call Termination

Calls between two endpoints can be either direct or routed via a gatekeeper. This scenario describes a direct connection where each endpoint is a point of entry and exit of a media flow.

The example in this section describes the procedure for placing a call between two endpoints A and B, each with an IP address on the same subnet.

1.3.1. Call Setup

Assuming a slow start connection procedure, establishing a call between two endpoints requires two TCP connections between the end points:

- One for the call setup
- One for capability exchange and call control

NOTE: H.225.0 version 2 also supports a fast start connection procedure where only one TCP connection is required. See *ITU-T Recommendation H.225.0* for more information.

The caller at endpoint A connects to the callee at endpoint B on a well-known port, port 1720, and sends the call *Setup* message as defined in the H.225.0 specification. The *Setup* message includes:

- Message type, in this case, *Setup*
- Bearer capability, which indicates the type of call, for example, audio only
- Called party number and address
- Calling party number and address
- Protocol Data Unit (PDU), which includes an identifier that indicates which version of H.225.0 should be used and other information.

When endpoint B receives the *Setup* message, it responds with one of the following messages:

- *Release Complete*
- *Alerting*
- *Connect*
- *Call Proceeding*

In this case, endpoint B responds with the *Alerting* message. Endpoint A must receive the *Alerting* message before its setup timer expires. After sending this message, the user at endpoint B must either accept or refuse the call with a predefined time period. When the user at endpoint B picks up the call, a *Connect* message is sent to endpoint A and the next phase of the call scenario, Capability Exchange, can begin.

1.3.2. Capability Exchange

Call control and capability exchange messages, as defined in the H.245 standard are sent on a second TCP connection. Endpoint A opens this connection on a dynamically allocated port at the endpoint B after receiving the address in one of the following H.225.0 messages:

- *Alerting*
- *Call Proceeding*
- *Connect*

This connection remains active for the entire duration of the call. The control channel is unique for each call between endpoints so that several different media streams can be present.

An H.245 *TerminalCapabilitySet* message that includes information about the codecs supported by that endpoint is sent from one endpoint to the other. Both endpoints send this message and wait for a reply which can be one of the following messages:

- *TerminalCapabilitySetAck* - Accept the remote endpoints capability
- *TerminalCapabilitySetReject* - Reject the remote endpoints capability

The two endpoints continue to exchange these messages until a capability set that is supported by both endpoints is agreed. When this occurs, the next phase of the call scenario, Call Initiation, can begin.

1.3.3. Call Initiation

Once the capability setup is agreed, endpoint A and B must set up the voice channels over which the voice data (media stream) will be exchanged.

To open a logical channel at endpoint B, endpoint A sends an H.245 *OpenLogicalChannel* message to endpoint B. This message specifies the type of data being sent, for example, the codec that will be used. For voice data, the message also includes the port number that endpoint B should use to send RTCP receiver reports. When endpoint B is ready to receive data, it sends an *OpenLogicalChannelAck* message to endpoint A. This message contains the port number on which endpoint A is to send RTP data and the port number on which endpoint A should send RTCP data.

Endpoint B repeats the process above to establish on which ports endpoint A will send both RTP data and RTCP reports. Once these ports have been identified, the next phase of the call scenario, Data Exchange, can begin.

1.3.4. Data Exchange

Endpoint A and endpoint B exchange information in RTP packets that carry the voice data. Periodically, during this exchange both sides send RTCP packets, which are used to monitor the quality of the data exchange. If endpoint A or endpoint B determines that the expected rate of exchange is being degraded due to line problems, H.323 provides capabilities to make adjustments. Once the data exchange has been completed, the next phase of the call scenario, Call Termination, can begin.

1.3.5. Call Termination

To terminate an H.323 call, one of the endpoints, for example, endpoint A hangs up. Endpoint A must send an H.245 *CloseLogicalChannel* message for each channel it has opened with endpoint B. Accordingly, endpoint B must reply to each of those messages with a *CloseLogicalChannelAck* message. When all the logical channels

are closed, endpoint A sends an H.245 *EndSessionCommand*, waits until it receives the same message from endpoint B, then closes the channel.

Both endpoint A and endpoint B then send an H.225.0 *ReleaseComplete* message over the call signalling channel, which closes that channel and ends the call.

1.4. H.323 Call Scenario Via a Gateway

While the call scenario described in Section 1.3, "Basic H.323 Call Scenario", on page 5 is useful for explaining the fundamentals of an H.323 call, is not a realistic call scenario. The IP addresses of both endpoints were defined to be known. Most Internet Service Providers (ISPs) allocate IP addresses to subscribers dynamically. This section describes the fundamentals of a more realistic example that involves a gateway.

A gateway provides a bridge between different technologies for example, an H.323 gateway (or IP gateway) provides a bridge between an IP network and the PSTN. Figure 3 shows a configuration that uses a gateway. User A is at a terminal, while user B is by a phone connected to the PSTN.

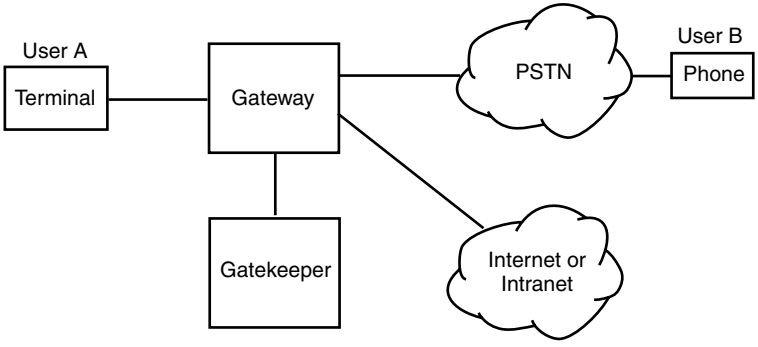


Figure 3. Basic H.323 Network with a Gateway

Figure 3 also shows a gatekeeper. The gatekeeper provides network services such as Registration, Admission and Status (RAS) and address mapping. When a gatekeeper is present, all endpoints managed by the gatekeeper must register with

the gatekeeper at startup. The gatekeeper tracks which endpoints are accepting calls. The gatekeeper can perform other functions also, such as redirecting calls. For example, if a user does not answer the phone, the gatekeeper may redirect the call to an answering machine.

The call scenario in this example involves the following phases:

- Establishing Contact with the Gatekeeper
- Requesting Permission to Call
- Call Signaling and Data Exchange
- Call Termination

1.4.1. Establishing Contact with the Gatekeeper

The user at endpoint A attempts to locate a gatekeeper by sending out a *Gatekeeper Request (GRQ)* message and waiting for a response. When it receives a *Gatekeeper Confirm (GCF)* message, the endpoint registers with the Gatekeeper by sending the *Registration Request (RRQ)* message and waiting for a *Registration Confirm (RCF)* message. If more than one gatekeeper responds, endpoint A chooses only one of the responding gatekeepers. The next phase of the call scenario, Requesting Permission to Call, can now begin.

1.4.2. Requesting Permission to Call

After registering with the gatekeeper, endpoint A must request permission from the gatekeeper to initiate the call. To do this, endpoint A sends an *Admission Request (ARQ)* message to the gatekeeper. This message includes information such as:

- A sequence number
- A gatekeeper assigned identifier
- The type of call, in this example, point-to-point
- The call model to use, either direct or gatekeeper-routed
- The destination address, in this case, the phone number of endpoint B
- An estimation of the amount of bandwidth required. This parameter can be adjusted later by a *Bandwidth Request (BRQ)* message to the gatekeeper.

If the gatekeeper allows the call to proceed, it sends an *Admission Confirm (ACF)* message to endpoint A. The *ACF* message includes the following information:

- The call model used
- The transport address and port to use for call signaling (in this example, the IP address of the gateway)
- The allowed bandwidth

All setup has now been completed and the next phase of the scenario, Call Signaling, can begin.

1.4.3. Call Signaling and Data Exchange

Endpoint A can now sends the *Setup* message to the gateway. Since the destination phone is connected to an analog line (the PSTN), the gateway goes off-hook and dials the phone number using dual tone multifrequency (DTMF) digits. The gatekeeper therefore is converting the H.225.0 signaling into the signaling present on the PSTN. Depending on the location of the gateway, the number dialed may need to be converted. For example, if the gateway is located in Europe, then the international dial prefix will be removed.

As soon as the gateway is notified by the PSTN that the phone at endpoint B is ringing, it sends the H.225.0 *Alerting* message as a response to endpoint A. As soon as the phone is picked up at endpoint B, the H.225.0 *Connect* message is sent to endpoint A. As part of the *Connect* message, a transport address that allows endpoint A to negotiate codecs and media streams with endpoint B is sent.

The H.225.0 and H.245 signaling used to negotiate capability, initiate and call, and exchange data are the same as that described in the basic H.323 call scenario. See Section 1.3.2, "Capability Exchange", on page 6, Section 1.3.3, "Call Initiation", on page 7, and Section 1.3.4, "Data Exchange", on page 7 for more information.

In this example the destination phone is analog, therefore, it requires the gateway to detect the ring, busy and connect conditions so it can respond appropriately.

1.4.4. Call Termination

As in the basic H.323 call scenario example, the endpoint that hangs up first needs to close all the channels that were open using the H.245 *CloseLogicalChannel* message. If the gateway terminates first, it sends an H.245 *EndSessionCommand*

message to endpoint A and waits for the same message from endpoint A. The gateway then closes the H.245 channel.

When all channels between endpoint A and the gateway are closed, each must send a *DisengageRequest (DRQ)* message to the gatekeeper. This message lets the gatekeeper know that the bandwidth is being released. The gatekeeper sends a *DisengageConfirm (DCF)* message to both endpoint A and the gateway.

1.5. References

The following publications provide related information.

- ITU-T Recommendation H.323 (09/98) - *Packet-based multimedia communications systems*
- ITU-T Recommendation H.245 (09/98) - *Control protocol for multimedia communication*
- ITU-T Recommendation H.225.0 (02/98) - *Call signaling protocols and media stream packetization for packet-based multimedia communications systems*
- RFC 1889, RTP: *A Transport Protocol for Real-Time Applications*, IETF Publication.
- Douskalis, Bill, Hewlett-Packard (Copyright 2000), *IP Telephony*, Prentice Hall PTR, Prentice-Hall, Inc.
- Black, Uyless (Copyright 2000), *Voice over IP*, Prentice Hall PTR, Prentice-Hall, Inc.
- *Introduction to Voice Over the Internet Protocol*, Paolo Galtieri, Applied Computing Technologies, Winter 2000

2. Using Global Call with IP Technology

Global Call provides a common call control interface that is independent of the underlying network interface technology. While Global Call is primarily concerned with call control, that is, call establishment and teardown, Global Call provides some additional capabilities to support applications that use IP technology.

Global Call support for IP technology includes:

- Call control capabilities for establishing calls over an IP network.
- Support for IP Media control by providing the ability to open and close IP Media channels for streaming.

2.1. Global Call Over IP Architecture with an Embedded Stack

Global Call supports a system configuration where the IP signaling stack is running as embedded software on an IP resource board (a DM3 board) and that board's network interface card (NIC) is used to access the IP network. The IP Media stream is also processed on the same IP resource board as the signaling stack.

NOTE: Global Call cannot be used for IP call control in a system configuration that uses a host-based stack, that is, the IP signaling stack is running on the host. However, the IP Media Library (for media resource management) can be used in such a configuration.

Figure 4 shows the Global Call over IP architecture in this context.

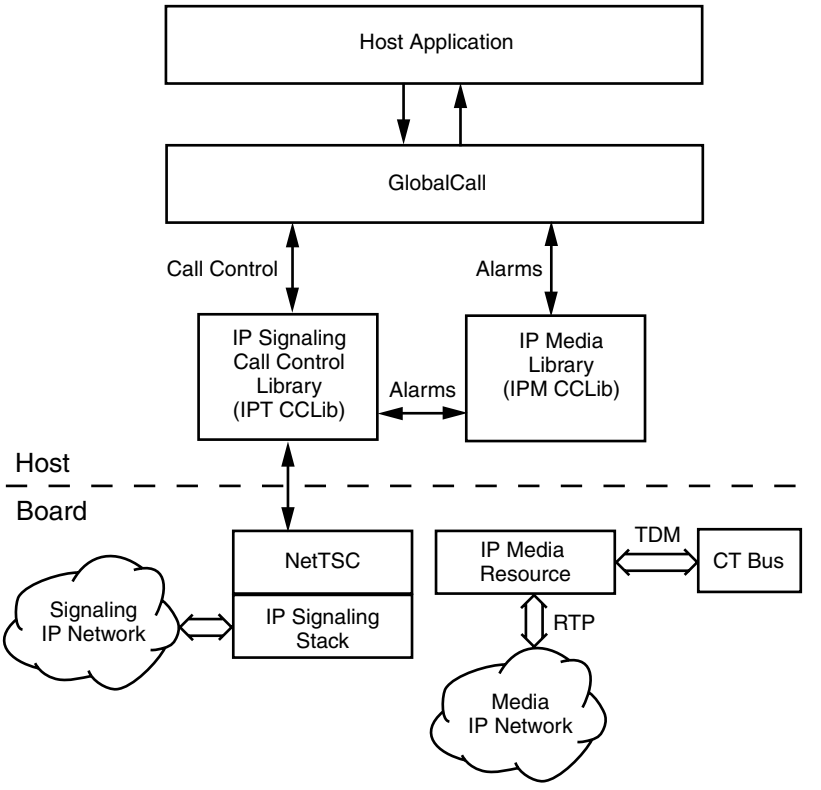


Figure 4. Global Call over IP Architecture Using an Embedded Stack

The role of each major component in the architecture is described in the sections following.

2.1.1. Host Application

The host application manages and monitors the IP telephony system operations. It performs the following tasks:

- Initializes Global Call

- Opens and closes IP line devices
- Opens and closes IP Media devices
- Configures IP Media and network devices (capability list, operation mode, etc.)
- Performs call control, including making calls, accepting calls, answering calls, dropping calls, releasing calls and processing call state events.
- Queries call and device information.
- Handles alarms and errors.

2.1.2. Global Call

Global Call hides technology and protocol-specific information from the host application and acts as an intermediary between the host application and the technology call control libraries. It performs the following tasks:

- Performs high-level call control using the underlying call control libraries.
- Maintains a generic call control state machine based on the function calls used by an application and call control library events.
- Collects and maintains data relating to resources.
- Collects and maintains alarm data.

2.1.3. IP Signaling Call Control Library (IPT CCLib)

The IP Signaling call control library (IPT CCLib) implements IP technology. It performs the following tasks:

- Controls the H.323 stack.
- Manages IP Media resources as required by the Global Call call state model and the IP signaling protocol model.
- Translates between the Global Call call model and IP signaling protocol model.
- Processes Global Call call control library interface commands.
- Generates call control library interface events.

2.1.4. IP Media Call Control Library (IPM CCLib)

The IP Media Call Control Library (IPM CCLib) performs the following tasks:

- Processes Global Call call control library interface commands for the opening, closing, and timeslot routing of media devices.
- Configures QoS thresholds.
- Translates QoS alarms to Global Call alarm events.

2.1.5. IP Media Resource

The IP Media Resource processes the IP Media stream. It performs the following tasks:

- Encodes PCM data from the TDM bus into IP packets sent to the internet.
- Decodes IP packets received from the Internet into PCM data transmitted to the TDM bus.
- Injects tones into the IP Media stream in response to requests from the IP Media Library. This is dependent on the setting of the **PrmDTMFXferMode** parameter in the *.config* file. See Section 4.5, "Configuring DTMF Handling", on page 58 for more information.
- Configures and reports Quality of Service (QoS) information to the IP Media stream.

2.2. Device Control

To simplify IP Media management by the host application and to provide a consistent look and feel with other Global Call technology call control libraries, the IP Signaling call control library (IPT CCLib) controls the IP Media functionality.

When using Global Call with IP technology, two types of devices are used:

- **IP Network Device** - Used for call control (call setup and tear down). The format of the device name is **iptBxTy**, where Bx is the logical board number and Ty is the logical channel number.
- **IP Media Device** - Used to control RTP streaming, monitoring Quality of Service (QoS) and sending and receiving DTMF digits. The format of the

device name is **ipmBxCy**, where Bx is the logical board number and Cy is the logical channel number.

When using Global Call with a DM3 board that uses an embedded call control stack, each IP Media device (ipmBxCy) is bound to a corresponding IP network device (iptBxTy). The IP network device has control over the IP Media device using the device handle.

The IP network device (iptBxTy) and the IP Media device (ipmBxCy) should be opened simultaneously, that is, in the same **gc_OpenEx( )** command. See Section 3.1.15, "gc_OpenEx( )", on page 30 for more information.

The IP Media device handle, which is required for managing Quality of Service (QoS) alarms for example, can be retrieved using the **gc_GetResourceH( )** function. See Section 4.7, "Setting and Retrieving Quality of Service (QoS) Thresholds", on page 60 for more information.

NOTE: If the IP network device (iptBxTy) and the IP Media device (ipmBxCy) are opened separately, Quality of Service (QoS) functions will fail.

Figure 5 shows the relationship between the various types of Global Call devices used in an IP environment.

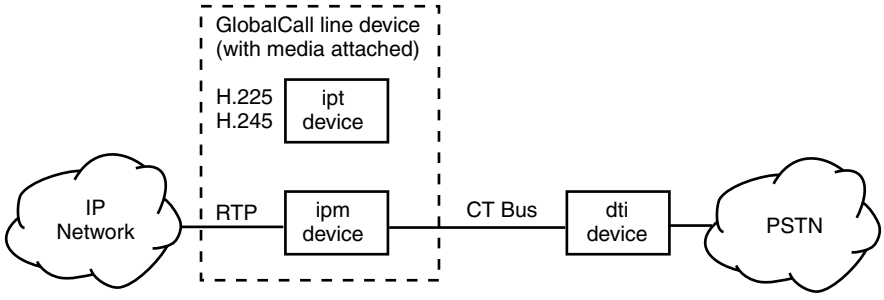


Figure 5. Global Call Devices in an IP Environment

NOTE: The board and channel/timeslot numbers in the ipt (ipt_BxCx) and ipm (ipm_BxTy) devices must be the same, that is, BxCx = BxTy.

3. Applying Global Call Functions to IP Technology

Certain Global Call functions have additional functionality or perform differently when used with IP technology. The generic function descriptions in the *Global Call API Software Reference* do not contain detailed information for any specific technology. Detailed information in terms of the additional functionality or the difference in performance of those functions when used with IP technology is contained in this chapter. The information provided in this guide therefore must be used in conjunction with the information presented in the *Global Call API Software Reference* to obtain the complete information when developing Global Call applications that use IP technology.

3.1. Function Variances

The following sections describe the details of using certain Global Call API functions in applications that use IP technology. See the *Global Call API Software Reference* for generic (technology-independent) descriptions of the Global Call API functions.

3.1.1. gc_AcceptCall( )

The **gc_AcceptCall( )** function is used to send the Q.931 ALERTING message to the originating end point. Also, the **rings** parameter is ignored.

3.1.2. gc_AnswerCall( )

The **rings** parameter is ignored.

Coders can be set in advance of using **gc_AnswerCall( )** by using one of the following functions:

- **gc_SetConfigData( )** - Enables the setting of system-wide parameters.

- **gc_SetUserInfo( )** - Enables the setting of parameters on a device basis when used with the **duration** parameter set to GC_ALLCALLS, or on a call basis when used with the **duration** parameter set to GC_SINGLECALL.

The following code example shows how to use:

- **gc_SetUserInfo( )** to set coder information before calls are answered using **gc_AnswerCall( )**.
- **gc_Extension( )** to retrieve the coder information once a connection is established and a GCEV_ANSWERED event is received.

```
/* Specifying coders before answering calls */

LINEDEV ldev;
CRN crn;
GC_PARM_BLK *target_datap
int i;

/* Define Coder */
IP_CAPABILITY a_DefaultCapability[2];

gc_OpenEx(&ldev, ":N_iptB1T1:M_ipmB1C1:P_H323_R", EV_SYNC, 0);

/* Set default coders for this ldev */
a_DefaultCapability[0].capability = GCCAP_AUDIO_g7231_6_3k;
a_DefaultCapability[0].direction = IP_CAP_DIR_LCLTRANSMIT;
a_DefaultCapability[0].type = GCCAPTYPE_AUDIO;
a_DefaultCapability[0].extra.audio.frames_per_pkt = 3;
a_DefaultCapability[0].extra.audio.VAD = GCPV_ENABLE;

a_DefaultCapability[1].capability = GCCAP_AUDIO_g711Alaw64k;
a_DefaultCapability[1].direction = IP_CAP_DIR_LCLTRANSMIT;
a_DefaultCapability[1].type = GCCAPTYPE_AUDIO;
a_DefaultCapability[1].extra.audio.frames_per_pkt = 10;
a_DefaultCapability[1].extra.audio.VAD = GCPV_DISABLE; /* Not applicable
                                                          for G.711 coders */

target_datap = NULL;

for (i = 0;i<2;i++) {
     gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
                             IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY),
                             &a_DefaultCapability[i]);
}

gc_SetUserInfo(GCTGT_GCLIB_CHAN, ldev, target_datap, GC_ALLCALLS);
gc_util_delete_parm(target_datap);

gc_WaitCall(ldev, EV_ASYNC);
```

```
/*... Receive GCEV_OFFERED ... */

gc_AnswerCall(crn, EV_ASYNC));
/* ... Receive GCEV_ANSWERED ... */

/* ... Check which coders have been used ...*/
GC_PARM_BLK *t_PrmBlkp = NULL;
GC_PARM_BLK *t_RetBlkp = NULL;
gc_util_insert_parm_val(&t_PrmBlkp, GCSET_CHAN_CAPABILITY,
                        IPPARM_LOCAL_CAPABILITY, 0, 0);
gc_Extension(GCTGT_GCLIB_CRN, crn, IPEXTID_GETINFO, t_PrmBlkp, &t_RetBlkp,
             EV_ASYNC);
gc_util_delete_parm(t_PrmBlkp);

/*.. Receive GCEV_EXTENSION*/
METAEVENT meta;
gc_GetMetaEvent(&meta);
EXTENSIONEVTBLK *extevtdatap = (EXTENSIONEVTBLK *)meta.extevtdatap;
GC_PARM_DATAP t_ParmData_p = NULL;

if (extevtdatap->ext_id == IPEXTID_GETINFO){
    while ((t_ParmData_p = gc_util_next_parm(&extevtdatap->parmblk,
            t_ParmData_p)) != NULL){
        if ((t_ParmData_p->set_ID == GCSET_CHAN_CAPABILITY) &&
            (t_ParmData_p->parm_ID == IPPARM_LOCAL_CAPABILITY) &&
            (t_ParmData_p->value_size != 0)){
            IP_CAPABILITY *t_Capability =
                                (IP_CAPABILITY *)(t_ParmData_p->value_buf);
            /* The direction will indicate RX or TX */
            if (t_Capability != NULL)
                printf("Coder Capability=%d,type=%d,direction=%d,fpp=%d,vad=%d",
                t_Capability->capability,
                t_Capability->type,
                t_Capability->direction,
                t_Capability->extra.audio.frames_per_pkt,
                t_Capability->extra.audio.VAD);
        }
    }
}
```

3.1.3. gc_AttachResource( )

When using Global Call with a DM3 board that uses an embedded call control stack, each IP Media device is preallocated (bound) to a specific network device, for example, iptB1T1 is bound to ipmB1C1, iptB1T2 is bound to ipmB1C2 etc. The **gc_AttachResource( )** function is redundant in this context, and therefore its use is **not** recommended.

3.1.4. gc_Detach( )

When using Global Call with a DM3 board that uses an embedded call control stack, each IP Media device is preallocated (bound) to a specific network device, for example, iptB1T1 is bound to ipmB1C1, iptB1T2 is bound to ipmB1C2 etc. The **gc_Detach( )** function is redundant in this context, and therefore its use is **not** recommended.

3.1.5. gc_DropCall( )

In addition to the drop call causes documented in the *Global Call API Software Reference*, the **cause** parameter can be any of the cause codes prefixed by IPEV_H225 or IPEC_Q931 specified in Table 29, "IP-Specific Event Cause Codes", on page 87.

NOTE: The **PrmStandardDisconnectReasonEnabled** in the *.config* file must be set to enable these codes.

3.1.6. gc_Extension( )

The **gc_Extension( )** function can be used for two purposes:

- To send protocol messages
- To retrieve call-related information

Table 1 shows the valid extension IDs and their purpose.

NOTES:
1. When using the IPEXTID_SENDMSG extension ID, the **gc_Extension( )** function must be called in synchronous mode, that is, the **mode** parameter must be EV_SYNC.
2. When using the IPEXTID_GETINFO extension ID, the **gc_Extension( )** function must be called in asynchronous mode, that is, the **mode** parameter must be EV_ASYNC.

Table 1. Valid Extension IDs for the gc_Extension( ) Function

Extension ID	Description
IPEXTID_SENDMSG	Used to send the following types of messages: • UII Messages (H.245) • Command Messages (H.245) • Facility Messages (Q.931) The parameter sets IDs supported are: • IPSET_MSG_H245 • IPSET_MSG_Q931 See Section 4.4, "Sending Protocol Messages", on page 53 for more information.
IPEXTID_GETINFO	Used to retrieve call-related information. See Section 4.3, "Retrieving Current Call-Related Information", on page 49 for more information.

3.1.7. gc_GetAlarmParm( )

The **gc_GetAlarmParm( )** function can be used to get QoS threshold values. The function parameter values in this context are:

- **linedev**: The media device handle, retrieved using the **gc_GetResourceH( )** function. See Section 4.7.1, "Retrieving the Media Device Handle", on page 61 for more information.
- **aso_id**: The alarm source object ID. Set to ALARM_SOURCE_ID_NETWORK_ID
- **ParmSetID**: Must be set to ParmSetID_qosthreshold_alarm.
- **alarm_parm_list**: A pointer to an ALARM_PARM_FIELD structure. The alarm_parm_number field is not used. The alarm_parm_data field is of type GC_PARM, which is a union. In this context, the type used is void *pstruct, and is cast as a pointer to an IPM_QOS_THRESHOLD_DATA structure, which contains the parameters representing threshold values. See the IPM_QOS_THRESHOLD_DATA structure in the *IP Media Library Software Reference* for more information.
- **mode**: Must be set to EV_SYNC.

There are two options for retrieving the values associated with QoS thresholds.

- To retrieve the values associated with all available thresholds, set the n_parms field in the ALARM_PARM_LIST structure (pointed to by the **alarm_parm_list** function parameter) to 0.
- To retrieve the values associated with specific thresholds, set the n_parms field in the ALARM_PARM_LIST structure to the number of thresholds (> 0) whose values are to be retrieved. In this case, you must also set the eQoStype field in the IPM_QOS_THRESHOLD_DATA structure corresponding to each threshold.

NOTE: Applications **must** include the *ipmlib.h* and *gcipmlib.h* header files before Global Call can be used to set or retrieve QoS threshold values.

3.1.8. gc_GetANI( )

The **gc_GetANI( )** function can be used to retrieve the IP address of the network interface card (NIC) of the calling party. When the function completes successfully, the **ani_buf** function parameter contains the address where the IP address of the calling party is stored.

3.1.9. gc_GetCallInfo( )

The **gc_GetCallInfo( )** function can be used to retrieve the calling number or the called number. The supported values of the **info_id** parameter are:

- DESTINATION_ADDRESS - the IP address of the called party (equivalent to DNIS)
- ORIGINATION_ADDRESS - the IP address of the calling party (equivalent to ANI)

3.1.10. gc_GetDNIS( )

The **gc_GetDNIS( )** function can be used to retrieve the IP address of the network interface card (NIC) of the called party. When the function completes successfully, the **dnis_buf** function parameter contains the address where the IP address of the called party is stored.

3.1.11. gc_GetResourceH( )

The **gc_GetResourceH( )** function can be used to retrieve the media device handle, which is required by GCAMS functions, such as, **gc_SetAlarmParm( )** and **gc_GetAlarmParm( )** to set and retrieve QoS threshold values. The function parameter values in this context are:

- **linedev** - the network device, that is, the Global Call line device retrieved by the **gc_OpenEx( )** function
- **resourcehp** - the address where the media device handle is stored when the function completes
- **resourcetype** - GC_MEDIADEVICE

NOTE: Applications **must** include the *ipmlib.h* and *gcipmlib.h* header files before Global Call can be used to set or retrieve QoS threshold values.

3.1.12. gc_GetXmitSlot( )

The **gc_GetXmitSlot( )** function can be used to get the transmit time slot information for an IP Media device. The format of the **gc_GetXmitSlot( )** function in this context is:

gc_GetXmitSlot(linedev, sctsinfop)

where,

- **linedev** is the Global Call line device handle for an IP device (that is, the handle returned by **gc_OpenEx( )** for a device with :N_iptBxTy in the **devicename** parameter).
- **sctsinfop** is a pointer to the transmit time slot information for the IP Media device (a pointer to a CT Bus time slot information structure).

See Section 4.6, “Call Routing”, on page 60 for related information.

3.1.13. gc_Listen( )

The **gc_Listen( )** function can be uses to have an IP Media device listen to another device (for example, a dtiBxTx device).

To compel an IP Media device to listen to another device, the format is:

gc_Listen(linedev, sctsinfop, EV_SYNC)

where,

- **linedev** is the Global Call line device handle for an IP device (that is, the handle returned by **gc_OpenEx( )** for a device with :N_iptBxTy in the **devicename** parameter).
- **sctsinfop** is a pointer to the transmit time slot information of the device to listen to. For a dtiBxTy device, this can be obtained by using **gc_GetXmitSlot( )** on a Global Call line device that has :N_dtiBxTy in the **devicename** parameter.
- **EV_SYNC** indicates synchronous mode.

See Section 4.6, "Call Routing", on page 60 for related information.

3.1.14. gc_MakeCall( )

Call parameters (see Table 2) can be specified when using the **gc_MakeCall( )** function. The parameters values specified are only valid for the duration of the current call. At the end of the current call, the default parameter values for the specific line device override these parameter values.

The **makecallp** parameter of the **gc_MakeCall( )** function is a pointer to the GC_MAKECALL_BLK structure. The GC_MAKECALL_BLK structure has a gclib field that points to a GCLIB_MAKECALL_BLK structure. The gclib.ext_datap field in turn points to a GC_PARM_BLK structure with a list of the parameters to be set as call values. Table 2 describes the parameters that can be accessed through the gclib.ext_datap pointer.

Table 2. Call Parameters Configurable using gc_MakeCall( )

Set ID	Parameter IDs
GCSET_CHAN_CAPABILITY	• IPPARM_LOCAL_CAPABILITY **Datatype**: IP_CAPABILITY. See Section 7.2, "IP_CAPABILITY", on page 80 for more information.

Table 2. Call Parameters Configurable using gc_MakeCall( )

Set ID	Parameter IDs
IPSET_CALLINFO See Section 6.2, "IPSET_CALLINFO Parameter Set", on page 73 for more information.	• IPPARM_DISPLAY **Datatype**: String. Null-terminated with a maximum size of 82 characters. • IPPARM_USERUSER_INFO **Datatype**: Uint8[ ]. Not null-terminated with a maximum size of 131 characters. • IPPARM_PHONELIST **Datatype**: String. Null-terminataed with a maximum size of 31 characters.
IPSET_NONSTANDARDDATA See Section 6.6, "IPSET_NONSTANDARDDATA Parameter Set", on page 75 for more information.	• IPPARM_NONSTANDARDDATA_OBJID **Datatype**: String • IPPARM_NONSTANDARDDATA_DATA **Datatype**: String

There are two options for setting the destination address when using the **gc_MakeCall( )** function:

- The destination address can be specified in the **numberstr** parameter of the **gc_MakeCall( )** function. The value of the **numberstr** parameter must indicate a type using one of the following prefixes:
  - TA: (IP address)
  - TEL: (telephone number)
  - NAME: (name)

  For example, a valid **numberstr** value is "TA:127.0.0.1". See Chapter 9, "Called and Calling Party Address List Format" for more information. If the destination address is specified in the **numberstr** parameter, the destination address should not be specified in the GCLIB_MAKECALL_BLK structure. Other fields in the GC_MAKECALL_BLK structure can still be used, for example, coder information.
- The destination address can be specified in the destination.address field in the GCLIB_MAKECALL_BLK structure. The prefixes, TA:, TEL:, or NAME: should **not** be used in this case. The destination.address_type field defines the type (IP address, name, telephone number) in the destination.address field. For example, if the destination.address field is "127.0.0.1", the

destination.address_type field must be GCADDRTYPE_IP. Other supported address types are:

- GCADDRTYPE_INTL - International telephone number
- GCADDRTYPE_NAT - National telephone number
- GCADDRTYPE_LOCAL - Local telephone number
- GCADDRTYPE_DOMAIN - Domain name
- GCADDRTYPE_URL - URL name
- GCADDRTYPE_EMAIL - email address

In this case, any address specified in the **numberstr** parameter of the **gc_MakeCall( )** function is ignored.

NOTE: When both methods above are used to specify the destination address, the address in the GCLIB_MAKECALL_BLK structure takes precedence.

In asynchronous mode, if the **timeout** parameter is set to 0, a call can be disconnected if either the underlying stack times out waiting for a response from the peer or, if the peer decides to reject the call. There are three timer parameters in the H.323 stack configuration file, *config.val*, that can be used to configure the stack timeout behavior:

- **Q931\responseTimeOut**: Maximum time in seconds to wait for the first response to a new call. If no response is received during this time, the Disconnect procedure is initiated. The default value is 50 sec.
- **Q931\connectTimeOut**: Maximum time in seconds to wait for the establishment of a new call, after receiving the first response to the call. If the call is not established during this time, the Disconnect procedure is initiated. The default value is 500 sec.
- **h245\timeout**: The maximum time in seconds to wait for the called party to acknowledge receipt of the capabilities it sent. The default value is 40 sec.

See the *DM3 IPLink User's Guide for Windows* for more information about the syntax of the *config.val* file.

The following code example shows how to:

- Create a GC_PARM_BLK with information on four supported coders, display information, a phone list and user-to-user information.
- Use the GC_PARM_BLK in a **gc_MakeCall( )** function.

NOTE: When using **gc_MakeCall( )** to set coder capabilities, only the coder capabilities in the transmit direction (IP_CAP_DIR_LCLTRANSMIT) can be specified. The function fails if attempting to specify the coder capabilities in the receive direction (IP_CAP_DIR_LCLRECEIVE).

```
/* Make an IP call on line device ldev */
void MakeIpCall(LINEDEV ldev)
{
    char * IpDisplay = "This is a Display"; /* display data */
    char * IpPhoneList= "003227124311";     /* phone list */
    char * IpDisplay = "This is a UUI";     /* user to user information string */
    char *paddr = "TA:127.0.0.1";           /* destination IP address */

    int rc = 0;
    const int MakeCallTimeout = 30;
    CRN crn;
    GC_MAKECALL_BLK gcmkbl;

    /* initialize GCLIB_MAKECALL_BLK structure */
    GCLIB_MAKECALL_BLK gclib_mkbl = {0};
    /* set to NULL to retrieve new parameter block from utility function */
    GC_PARM_BLK *target_datap = NULL;
    gcmkbl.cclib = NULL;  /* CCLIB pointer unused */
    gcmkbl.gclib = &gclib_mkbl;
    /* initialize IP_CAPABILITY structure */
    IP_CAPABILITY t_Capability = {0};

    /* configure a GC_PARM_BLK with four coders, display, phone list
       and UUI message: */
    /* specify and insert first capability parameter data for G.7231 coder */
    t_Capability.type = GCCAPTYPE_AUDIO;
    t_Capability.direction = IP_CAP_DIR_LCLTRANSMIT;
    t_Capability.extra.audio.VAD = GCPV_DISABLE;
    t_Capability.extra.audio.frames_per_pkt = 1;
    t_Capability.capability = GCCAP_AUDIO_g7231_6_3k;

    rc = gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
         IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &t_Capability);


    /* specify and insert second capability parameter data for G.7229AnnexA
       coder */
    /* changing only frames per pkt and the coder type from first capability: */
    t_Capability.extra.audio.frames_per_pkt = 3;
    t_Capability.capability = GCCAP_AUDIO_g729AnnexA;

    rc = gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
         IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &t_Capability);

    /* specify and insert 3rd capability parameter data for G.711Alaw 64kbit
```

```
        coder */
    /* changing only frames per pkt and the coder type from first capability: */
    t_Capability.capability = GCCAP_AUDIO_g711Alaw64k;
    t_Capability.extra.audio.frames_per_pkt = 10;
        /* For G.711 use frame size (ms) here, frames per packet fixed at 1 fpp */

    rc = gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
         IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &t_Capability);

    /* specify and insert fourth capability parameter data for G.711 Ulaw
       64kbit coder */
    /* changing only the coder type from previous capability */
    t_Capability.capability = GCCAP_AUDIO_g711Ulaw64k;

    rc = gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
         IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &t_Capability);

    /* insert display string */
    rc = gc_util_insert_parm_ref(&target_datap, IPSET_CALLINFO, IPPARM_DISPLAY,
         strlen(IpDisplay)+1, IpDisplay);

    /* insert phone list */
    rc = gc_util_insert_parm_ref(&target_datap, IPSET_CALLINFO, IPPARM_PHONELIST,
         strlen(IpPhoneList)+1, IpPhoneList);

    /* insert user to user information */
    rc = gc_util_insert_parm_ref(&target_datap, IPSET_CALLINFO,
         IPPARM_USERUSER_INFO, strlen(IpUUI)+1, IpUUI);

    if (rc == 0) {
        gclib_mkbl.ext_datap = target_datap;
        retcode = gc_MakeCall(ldev, &crn, paddr, &gcmkbl, MakeCallTimeout,
                  EV_ASYNC);
        /* deallocate GC_PARM_BLK pointer */
        gc_util_delete_parm(target_datap);
    }
}
```

3.1.15. gc_OpenEx( )

The format of the fields used to specify the **devicename** parameter is as follows:

```
:P_H323_NTSC:N_iptBxTy:M_ipmBxCy:V_dxxxBwCz
```

The prefixes (P_, N_, M_, and V_) are used for parsing purposes. These fields may appear in any order. The conventions described below allow the Global Call API to map subsequent calls made on specific line devices or CRNs to interface-specific

libraries. The fields within the **devicename** parameter must each begin with a colon.

The meaning of each field in the **devicename** parameter is as follows:

- **P_H323_NTSC**: Specifies that Global Call should use the H.323 embedded stack protocol. This field is mandatory.
- **N_iptBxTy**: Specifies the name of the IP network device. Bx and Ty represents the logical board and logical channel assigned by the user. When using Global Call with a DM3 board that uses an embedded stack, the range of logical channel numbers depends on the type of PSTN interface on the board (E-1 or T-1) and the number spans. For example, for a board with an E-1 interface and one span, the range of IP logical channels is 1 to 30. For a board with an E-1 interface and two spans, the range of IP logical channels is 1 to 60. When specifying an IP network device, it is recommended that an IP Media channel (M_ipmBxCx) be specified also.
- **M_ipmBxCy**: Specifies the name of the IP Media channel to be associated with the IP network device being opened. This field should be specified when the IP network device (N_iptBxTy) field is specified.
- **V_dxxxBwCz**: Specifies a voice resource. Bw and Cz are the voice board and channel number respectively. This field is optional.

  NOTE: The voice device should only be specified when using firmware downloads that support exportable voice resources, that is, the *.pcd* file for the firmware download has "**_evr_**" in the file name. When using a firmware download that does not support exportable voice resources, an error is generated if the voice device is specified since the voice device cannot be automatically routed.

The **gc_OpenEx( )** function can also be used to open a board device. This function is useful since it allows the use of the **ATDV_SUBDEVS( )** function to retrieve the number of subdevices for the board device. In this context, the **devicename** parameter has the following format:

```
:P_H323_NTSC:N_iptBx
```

In other technologies, a board device can be used for alarms. However, the use of an **iptBx** board device for alarms is not supported.

NOTES: 1. When using Global Call with a DM3 board that uses an embedded stack, each IP Media device is preallocated (bound) to a specific network device, for example, iptB1T1 is bound to ipmB1C1, iptB1T2 is bound to ipmB1C2, etc.

2. The IP address corresponding to the network interface is configured at firmware download time. For Windows systems, this is done using the Dialogic Configuration Manager (DCM).

3.1.16. gc_SetConfigData( )

The **gc_SetConfigData( )** function enables the setting of coder information for all devices on a board. The new coder settings apply only to devices that will be opened in the future, and do not override the coder settings of devices that are already open. The coder settings can be overridden by specifying coder information in the **gc_SetUserInfo( )** function (device level) or the **gc_MakeCall( )** function (call level).

NOTE: Coder information must be specified for a device before making a call (**gc_MakeCall( )**) or answering a call (**gc_AnswerCall( )**) on that device.

When using the **gc_SetConfigData( )** function for this purpose, use the following function parameter values:

- **target_type** - GCTGT_PROTOCOL_SYSTEM
- **target_id** - The H.323 protocol ID. Since the protocol is dynamically assigned by Global Call, the **gc_QueryConfigData( )** function must be used to obtain the correct protocol ID value. When using the **gc_QueryConfigData( )** function, use the following parameter values:
  - **target_type** - GCTGT_GCLIB_SYSTEM
  - **target_id** - 0
  - **source_datap** - H323_NTSC (the protocol name)
  - **query_id** - GCQUERY_PROTOCOL_NAME_TO_ID
  - **response_datap** - A pointer to the protocol ID.
- **target_datap** - A pointer to a GC_PARM_BLKP structure that contains the parameters to be configured. Table 3 describes the set ID and parameter ID values that can be used in this context.

Table 3. Parameters Configurable using gc_SetConfigData( )

Set ID	Parameter IDs
GCSET_CHAN_CAPABILITY	• IPPARM_LOCAL_CAPABILITY **Datatype**: IP_CAPABILITY. See Section 7.2, "IP_CAPABILITY", on page 80 for more information.

3.1.17. gc_SetAlarmParm( )

The **gc_SetAlarmParm( )** function can be used to set QoS threshold values. The function parameter values in this context are:

- **linedev**: The media device handle, retrieved using the **gc_GetResourceH( )** function. See Section 4.7.1, "Retrieving the Media Device Handle", on page 61 for more information.
- **aso_id**: The alarm source object ID. Set to ALARM_SOURCE_ID_NETWORK_ID
- **ParmSetID**: Must be set to ParmSetID_qosthreshold_alarm.
- **alarm_parm_list**: A pointer to an ALARM_PARM_FIELD structure. The alarm_parm_number field is not used. The alarm_parm_data field is of type GC_PARM, which is a union. In this context, the type used is void *pstruct, and is cast as a pointer to an IPM_QOS_THRESHOLD_DATA structure, which contains the parameters representing threshold values. See the IPM_QOS_THRESHOLD_DATA structure in the *IP Media Library Software Reference* for more information.
- **mode**: Must be set to EV_SYNC.

NOTE: Applications **must** include the *ipmlib.h* and *gcipmlib.h* header files before Global Call can be used to set or retrieve QoS threshold values.

3.1.18. gc_SetUserInfo( )

The **gc_SetUserInfo( )** function can be used to:

- Set call values for the duration of a single call
- Set call values for all calls on the specified line device

Setting Call Parameters for a Single Call

The pertinent function parameter values in this context are:

- **target_type** - GCTGT_GCLIB_CHAN
- **target_id** - the line device
- **duration** - GC_SINGLECALL
- **infoparmblkp** - a pointer to a GC_PARM_BLK with a list of parameters to be set for the line device. Table 4 describes the set ID and parameter ID values that can be used in this context and indicates when the data is used.

Table 4. Call Parameters Configurable using gc_SetUserInfo( )

Set ID	Parameter IDs	When Used*
GCSET_CHAN_CAPABILITY	IPPARM_LOCAL_CAPABILITY **Datatype**: IP_CAPABILITY. See Section 7.2, "IP_CAPABILITY", on page 80 for more information.	gc_AnswerCall( ) gc_MakeCall( )
IPSET_CALLINFO See Section 6.2, "IPSET_CALLINFO Parameter Set", on page 73 for more information.	IPPARM_DISPLAY **Datatype**: String	gc_AnswerCall( ) gc_MakeCall( )
	IPPARM_USERUSER_INFO **Datatype**: Uint8[ ]	gc_MakeCall( )
	IPPARM_PHONELIST **Datatype**: String	gc_MakeCall( ).
IPSET_NONSTANDARDDATA See Section 6.6, "IPSET_NONSTANDARDDATA Parameter Set", on page 75 for more information.	IPPARM_NONSTANDARDDATA_OBJID **Datatype**: String	gc_AnswerCall( ) gc_MakeCall( ) gc_DropCall( )
	IPPARM_NONSTANDARDDATA_DATA **Datatype**: String	gc_AnswerCall( ) gc_MakeCall( ) gc_DropCall( )
* Information can be set in any state but it is only used in certain states. If information is set after it can be used, and the duration parameter is set to GC_SINGLECALL, the information settings are ignored.		

Setting Call Parameters for a Line Device

When the **duration** parameter is set to GC_ALLCALLS, the new call values become the default values for the line device and are used for all subsequent calls on that device. The pertinent function parameter values in this context are:

- **target_type** - GCTGT_GCLIB_CHAN
- **target_id** - the line device
- **duration** - GC_ALLCALLS
- **infoparmblkp** - a pointer to a GC_PARM_BLK with the coder information to be set for the line device. Table 5 describes the set ID and parameter ID values that can be used in this context.

Table 5. Line Device Parameters Configurable using gc_SetUserInfo( )

Set ID	Parameter IDs
GCSET_CHAN_CAPABILITY	• IPPARM_LOCAL_CAPABILITY **Datatype**: IP_CAPABILITY. See Section 7.2, "IP_CAPABILITY", on page 80 for more information.

3.1.19. gc_UnListen( )

When the **gc_UnListen( )** function is used to stop an IP Media device listening to a digital network interface device, the **linedev** parameter must be the line device handle returned by **gc_OpenEx( )** when:

- the network (N_iptBxTy) and media (M_ipmBxCy) devices are specified in the **devicename** parameter
- a network (N_iptBxTy) device that has a media device attached is specified in the **devicename** parameter

3.2. Supported Global Call Functions

The following is a full list of the Global Call functions supported when using Global Call with IP technology:

- **gc_AcceptCall( )** - See Section 3.1.1, "gc_AcceptCall( )", on page 19 for variances.

- **gc_AnswerCall( )** - See Section 3.1.2, "gc_AnswerCall( )", on page 19 for variances.
- **gc_AttachResource( )** - Supported, but not used. See Section 3.1.3, "gc_AttachResource( )", on page 21.
- **gc_CCLibIDToName( )**
- **gc_CCLibNameToID( )**
- **gc_CCLibStatus( )**
- **gc_CCLibStatusAll( )**
- **gc_CCLibStatusEx( )**
- **gc_Close( )**
- **gc_CRN2LineDev( )**
- **gc_Detach( )** - Supported, but not used. See Section 3.1.4, "gc_Detach( )", on page 22.
- **gc_DropCall( )** - See Section 3.1.5, "gc_DropCall( )", on page 22.
- **gc_ErrorInfo( )**
- **gc_ErrorValue( )**
- **gc_Extension( )** - See Section 3.1.6, "gc_Extension( )", on page 22 for variances.
- **gc_GetANI( )** - See Section 3.1.7, "gc_GetAlarmParm( )", on page 23 for variances.
- **gc_GetCallInfo( )** - See Section 3.1.9, "gc_GetCallInfo( )", on page 24 for variances.
- **gc_GetCallState( )**
- **gc_GetCRN( )**
- **gc_GetCTInfo( )**
- **gc_GetDNIS( )** - See Section 3.1.10, "gc_GetDNIS( )", on page 24 for variances.
- **gc_GetLineDev( )**
- **gc_GetLineDevState( )**
- **gc_GetMetaEvent( )**
- **gc_GetMetaEventEx( )**
- **gc_GetNetworkH( )**

- **gc_GetResourceH( )** - See Section 3.1.11, “gc_GetResourceH( )”, on page 25 for variances.
- **gc_GetUsrAttr( )**
- **gc_GetVer( )**
- **gc_GetVoiceH( )**
- **gc_GetXmitSlot( )** - See Section 3.1.12, “gc_GetXmitSlot( )”, on page 25 for variances.
- **gc_Listen( )** - See Section 3.1.13, “gc_Listen( )”, on page 25 for variances.
- **gc_MakeCall( )** - See Section 3.1.14, “gc_MakeCall( )”, on page 26 for variances.
- **gc_Open( )**
- **gc_OpenEx( )** - See Section 3.1.15, “gc_OpenEx( )”, on page 30 for variances.
- **gc_ReleaseCall( )**
- **gc_ReleaseCallEx( )**
- **gc_ResetLineDev( )**
- **gc_ResultInfo( )**
- **gc_ResultMsg( )**
- **gc_ResultValue( )**
- **gc_SetCallingNum( )**
- **gc_SetConfigData( )** - See Section 3.1.16, “gc_SetConfigData( )”, on page 32 for variances.
- **gc_SetEvtMsk( )**
- **gc_SetUsrAttr( )**
- **gc_SetUserInfo( )** - See Section 3.1.18, “gc_SetUserInfo( )”, on page 33 for variances.
- **gc_Start( )**
- **gc_Stop( )**
- **gc_UnListen( )**
- **gc_util_parm_ref( )**
- **gc_util_parm_val( )**
- **gc_util_find_parm( )**
- **gc_util_next_parm( )**

- **gc_util_delete_parm_blk( )**
- **gc_WaitCall( )**

See the *Global Call API Software Reference* for more information about Global Call functions.

3.3. Supported Global Call Call States

The IP call control library supports all call states except the following:

- GCST_CALLROUTING
- GCST_DETECTED
- GCST_DIALING
- GCST_GETMOREINFO
- GCST_ONHOLD
- GCST_ONHOLDPENDINGTRANSFER
- GCST_SENDMOREINFO

See the *Global Call Application Developer's Guide* for more information about call state models.

3.4. Supported Global Call Events

The following Global Call events are supported when using Global Call with IP technology:

- GCEV_ACCEPT
- GCEV_ANSWERED
- GCEV_ALERTING
- GCEV_BLOCKED
- GCEV_CONNECTED
- GCEV_DISCONNECTED
- GCEV_DROPCALL
- GCEV_FATALERROR
- GCEV_OFFERED

- GCEV_OPENEX
- GCEV_OPENEX_FAIL
- GCEV_PROCEEDING
- GCEV_RELEASECALL
- GCEV_RESETLINEDEV
- GCEV_TASKFAIL
- GCEV_UNBLOCKED

See the *Global Call API Software Reference* for more information about Global Call events.

4. Using Global Call for IP-Specific Tasks

This chapter describes how to use Global Call to perform certain tasks in an IP environment. These tasks include:

- Using Fast and Slow Start Connections
- Setting Call-Related Information
- Retrieving Current Call-Related Information
- Sending Protocol Messages
- Configuring DTMF Handling
- Call Routing
- Setting and Retrieving Quality of Service (QoS) Thresholds

Each task is described in more detail in the following sections.

4.1. Using Fast and Slow Start Connections

An application can use one of two methods to establish a connection with an end point, fast start connection and slow start connection. Fast start connection allows the system to setup a call without the use of the H.245 control channel. If the remote side does not support Fast Start, then a slow start connection is established.

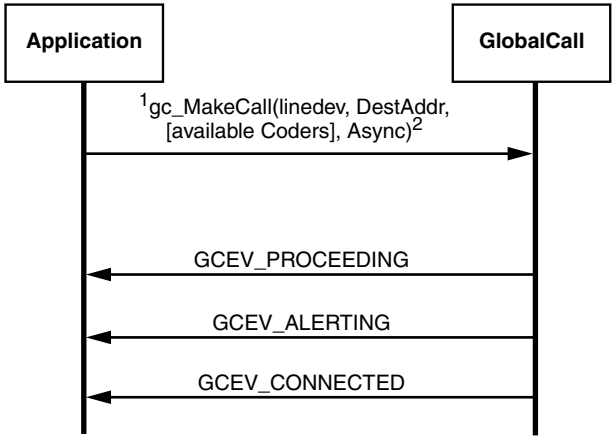
Fast start connection is preferable to slow start connection because fewer network round trips are required to set up a call and the local exchange can generate messages when circumstances prevent a connection to the end point.

NOTE: The connection mode (fast start connection or slow start connection) is specified by setting the value of the **PrmForceSlowStart** parameter (0=FastStart - default, 1=SlowStart) in the *.config* file. The FCDGEN utility is used to convert the *.config* file to a *.fcd* file that is then used by the downloader when downloading firmware to the board. The connection mode cannot be set dynamically at run time.

When using Global Call with a DM3 board that uses an embedded stack, the scenarios for fast start connection and slow start connection are the same.

4.1.1. Establishing a Connection as Viewed from the Calling Side

Figure 6 shows how a connection is established as viewed from the calling side.



Note: Makecall data is optional; defaults will be used if makecall data is not specified.

Figure 6. Establishing a Connection as Viewed from the Calling Side

4.1.2. Establishing a Connection as Viewed from the Called Side

Figure 7 shows how a connection is established as viewed from the called side.

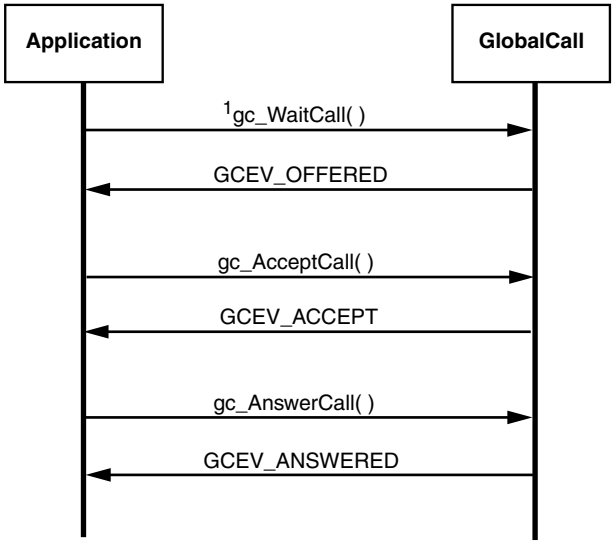


Figure 7. Establishing a Connection as Viewed from the Called Side

4.2. Setting Call-Related Information

Table 6 summarizes the types of information that can be specified, the corresponding set IDs and parameter IDs used to set the information and the functions that can be used to set the information.

Table 6. Summary of Call-Related Information that can be Set

Type of Information	Set ID and Parameter IDs	Functions Used to Set Information
Coder Information	GCSET_CHAN_CAPABILITY • IPPARM_LOCAL_CAPABILITY	gc_SetConfigData( ) gc_SetUserInfo( )* gc_SetUserInfo( )**
Display Information	IPSET_CALLINFO • IPPARM_DISPLAY	gc_MakeCall( ) gc_SetUserInfo( )**
User to User Information	IPSET_CALLINFO • IPPARM_USERUSER_INFO	gc_MakeCall( ) gc_SetUserInfo( )**
Phone List	IPSET_CALLINFO • IPPARM_PHONELIST	gc_MakeCall( ) gc_SetUserInfo( )**
Nonstandard Object ID	IPSET_NONSTANDARDDATA • IPPARM_NONSTANDARDDATA_OBJID	gc_MakeCall( ) gc_SetUserInfo( )**
Nonstandard Data	IPSET_NONSTANDARDDATA • IPPARM_NONSTANDARDDATA_DATA	gc_MakeCall( ) gc_SetUserInfo( )**

* When the **duration** parameter is set to GC_ALLCALLS, the settings apply on a line device basis.
** When the **duration** parameter is set to GC_SINGLECALL, the settings apply on a call basis.

4.2.1. Setting the Values of System-Wide Parameters

Use the **gc_SetConfigData( )** function to configure coder information. The values set by the **gc_SetConfigData( )** function are used by the call control library as default values for each line device that is subsequently opened. The coder settings apply only to devices that will be opened in the future, and do not override the coder settings of devices that are already open.

See Section 3.1.16, “gc_SetConfigData( )”, on page 32 for more information about the values of function parameters to set in this context.

4.2.2. Setting the Values of Line Device Parameters

The **gc_SetUserInfo( )** function can be used to set the values of line device parameters. The values set by **gc_SetUserInfo( )** become the new default values for the specified line device and are used by all subsequent calls on that device.

See Section 3.1.18, “gc_SetUserInfo( )”, on page 33 for more information about the values of function parameters to set in this context.

4.2.3. Setting the Values of Call Parameters

There are two ways to set the values of call parameters:

- Using the **gc_SetUserInfo( )** function
- Using the **gc_MakeCall( )** function

Using gc_SetUserInfo( ) to Set Call Parameter Values

Use the **gc_SetUserInfo( )** function to set information element values for a single incoming call. At the end of the call, the values set as default values for the specified line device override these values. This is useful since the **gc_AnswerCall( )** function does not have a parameter to specify a GC_PARM_BLK.

If a **gc_MakeCall( )** function is issued after the **gc_SetUserInfo( )**, the values specified in the **gc_MakeCall( )** function override the values specified by the **gc_SetUserInfo( )** function.

See Section 3.1.18, “gc_SetUserInfo( )”, on page 33 for more information about the values of function parameters to set in this context.

Using gc_MakeCall( ) to Set Call Parameter Values

Use the **gc_MakeCall( )** function to set the values of parameters for a call. The values set are only valid for the duration of the current call. At the end of the call,

the values set as default values for the specified line device override the values specified by the **gc_MakeCall( )** function.

See Section 3.1.14, "gc_MakeCall( )", on page 26 for more information about the values of function parameters to set in this context.

4.2.4. Setting and Retrieving Disconnect Cause or Reason Values

Use the **cause** parameter in the **gc_DropCall( )** function to specify a disconnect reason/cause to be sent to the remote end point.

Use the **gc_ResultInfo( )** function to get the reason/cause of a GCEV_DISCONNECTED event. This reason/cause could be sent from the remote end point or it could be the result of an internal error.

IP-specific reason/cause values are specified in the IPCAUSE_Values enumerator defined in the *gcip_defs.h* header file.

NOTE: The **PrmStandardDisconnectReasonEnabled** parameter (0x1e40) in the FCD file must be set to enable the reporting of these cause values. Failing to set this value results in the cause always being "Normal Clearing".

The following conversion macros are provided to translate reason/cause values defined by the Q.931 and H.225 standards to equivalent Global Call IP reason/cause values and vice-versa:

- H2250_REASON_TO_IPTGC_REASON(H225Reason, RetVal)
- IPTGC_REASON_TO_H2250_REASON(IPTGCReason, RetVal)
- Q931_CAUSE_TO_IPTGC_CAUSE(Q931Cause, RetVal)
- IPTGC_CAUSE_TO_Q931_CAUSE(IPTGCCause, RetVal)

4.2.5. Setting Coder Information

Terminal capabilities are exchanged as part of the H.245 session used during call establishment. The terminal capabilities are sent to the remote side as notification of coder supported.

The terminal capability list for the system is set in the *config.val* file and is configured at firmware download time. See the *IPLink User's Guide* for more information.

Table 7 shows the coders that are supported when using Global Call with a DM3 board that uses an embedded stack.

Table 7. Coders Supported by the Global Call API

Coder and Rate	Global Call # Define	Frames Per Packet (fpp) or Frame Size (ms)	VAD Support
G.711 ALaw	GCCAP_AUDIO_g711Alaw64k	Frame Size: 10, 20, or 30 ms (Frames Per Packet: Fixed at 1 fpp)	Not applicable
G.711 ULaw	GCCAP_AUDIO_g711Ulaw64k	Frame Size: 10, 20, or 30 ms (Frames Per Packet: Fixed at 1 fpp)	Not applicable
G.723.1 5.3 Kbps	GCCAP_AUDIO_g7231_5_3k	Frames Per Packet: 1, 2 or 3 fpp (Frame Size: Fixed at 30 ms)	Enabled (default) or disabled
G.723.1, 6.3 Kbps	GCCAP_AUDIO_g7231_6_3k	Frames Per Packet: 1, 2 or 3 fpp (Frame Size: Fixed at 30 ms)	Enabled (default) or disabled
G.729A	GCCAP_AUDIO_g729AnnexA	Frames Per Packet: 3 or 4 fpp (Frame Size: Fixed at 10 ms)	Not applicable
G.729A with AnnexB	GCCAP_AUDIO_g729AnnexA wAnnexB	Frames Per Packet: 3 or 4 fpp (Frame Size: Fixed at 10 ms)	Not applicable (VAD is supported implicitly)
GSM FR	GCCAP_AUDIO_gsmFullRate	Frames Per Packet: 1, 2 and 3 fpp (Frame Size: Fixed at 20 ms)	Disabled (default) or Enabled
Note: For G.711 coders, the frame size value (not the frames per packet value) is specified in the frames_per_pkt field of the IP_AUDIO_CAPABILITY structure. See Section 7.1, "IP_AUDIO_CAPABILITY", on page 79 for more information.			

Coder information can be set in the following ways:

- On a system wide basis using **gc_SetConfigData( )**.

- On a line device basis using **gc_SetUserInfo( )** with a **duration** parameter value of GC_ALLCALLS.
- On a call basis using **gc_MakeCall( )** or **gc_SetUserInfo( )** with a **duration** parameter value of GC_SINGLECALL.

In each case, the following parameter set ID and parameter IDs are used to set up a GC_PARM_BLK containing the coder information:

- GCSET_CHAN_CAPABILITY
  - IPPARM_LOCAL_CAPABILITY - a parameter of type IP_CAPABILITY

Possible values for fields in the IP_CAPABILITY structure are:

- capability - One of the following:
  - GCCAP_AUDIO_g711Alaw64k
  - GCCAP_AUDIO_g711Ulaw64k
  - GCCAP_AUDIO_g7231_5_3k (at 5.3 kbits/s)
  - GCCAP_AUDIO_g7231_6_3k (at 6.3 kbits/s)
  - GCCAP_AUDIO_g729AnnexA
  - GCCAP_AUDIO_g729AnnexAwAnnexB
  - GCCAP_AUDIO_gsmFullRate
- type - GCCAPTYPE_AUDIO
- direction - One of the following
  - IP_CAP_DIR_LCLTRANSMIT - transmit capability
  - IP_CAP_DIR_LCLRECEIVE - receive capability
- payload_type - Not supported.
- extra - Must be of type IP_AUDIO_CAPABILITY.
  - extra.frames_per_packet - The number of frames per packet.

  NOTE: For G.711 coders, this field is the frame size (in ms), which must be one of the following values only: 10, 20, or 30.
  - extra.VAD - 0 (disabled) or 1 (enabled)

See Section 7.2, "IP_CAPABILITY", on page 80 for more information.

The application can retrieve information about the transmit coder that the remote side used (see Section 4.3, “Retrieving Current Call-Related Information”, on page 49), but the application cannot get a list of supported coders once a call has been answered.

4.2.6. Specifying Nonstandard Data for H.245 Messages

Use the **gc_SetUserInfo( )** function with a **duration** parameter set to GC_SINGLECALL to set non-standard data. If the **duration** parameter is set to GC_ALLCALLS, the function will fail.

The following parameter set ID and parameter IDs should be used when setting up the GC_PARM_BLK pointed by the **infoparmblkp** function parameter:

- IPSET_NONSTANDARDDATA
  - IPPARM_NONSTANDARDDATA_OBJID
  - IPPARM_NONSTANDARDDATA_DATA

See Section 6.6, “IPSET_NONSTANDARDDATA Parameter Set”, on page 75 for more information.

4.3. Retrieving Current Call-Related Information

To support large numbers of channels, the call control library must perform all operations in *asynchronous* mode. To support this, an extension function variant allows the retrieval of a parameter as an asynchronous operation.

The retrieval of call-related information is a four step process:

- Set up a GC_PARM_BLK that identifies which information is to be retrieved. The GC_PARM_BLK includes GC_PARM_DATA blocks. The GC_PARM_DATA blocks specify only the Set_ID and Parm_ID fields, that is, the value_size field is set to 0. The list of GC_PARM_DATA blocks indicate to the call control library the parameters to be retrieved.
- Use the **gc_Extension( )** to request the data. The **ext_id** function parameter (extension ID) should be set to IPEXTID_GETINFO, the **parmblkp** function parameter should point to the GC_PARM_BLK set up in step 1, and the **mode** function parameter should be set to EV_ASYNC (asynchronous).

- A GCEV_EXTENSION event is generated in response to the **gc_Extension( )** request. The GCEV_EXTENSION event has a GC_PARM_BLK associated with it that contains the requested information.
- Extract the information from the GC_PARM_BLK associated with the GCEV_EXTENSION event. In this case, the GC_PARM_BLK contains real data, that is, the value_size field is not 0, but includes the size of the data following for each parameter requested.

Table 8 shows the parameters that can be retrieved and when the information should be retrieved.

Table 8. Retrievable Call Information

Parameter	Set ID	Parameter ID	When Information Can Be Retrieved	Datatype in value_buf Field (See Note Below)
RTCP Information	GCSET_ CHAN_ CAPABILITY	IPPARM_ RTCPINFO	In any state but has no meaning before the Connected state. Contains 0's prior to the Connected state.	RTCPINFO*
Coder Information	IPSET_ CALLINFO	IPPARM_ LOCAL_ CAPABILITY	After the Connected or Answered state.	IP_ CAPABILITY*
Display Information	IPSET_ CALLINFO	IPPARM_ DISPLAY	Any state after Offered or Proceding.	char*
User to User Information	IPSET_ CALLINFO	IPPARM_ USERUSER_ INFO	Any state after Offered or Proceding.	char*
Call Duration	IPSET_ CALLINFO	IPPARM_ CALL DURATION	Only after **gc_DropCall( )** but before **gc_Release( )**	int*
Phone List	IPSET_ CALLINFO	IPPARM_ PHONELIST	Any state after Offered or Proceding.	char*

Table 8. Retrievable Call Information

Parameter	Set ID	Parameter ID	When Information Can Be Retrieved	Datatype in value_buf Field (See Note Below)
Nonstandard Object ID	IPSET_ CALLINFO	IPPARM_ NONSTANDARD DATA_OBJID	See Section 4.3.1, "Retrieving Nonstandard Data From Protocol Messages", on page 52 for more information.	char*
Nonstandard Data	IPSET_ CALLINFO	IPPARM_ NONSTANDARD DATA_DATA	See Section 4.3.1, "Retrieving Nonstandard Data From Protocol Messages", on page 52 for more information.	Uint[8]
Vendor Product ID	IPSET_ VENDORINFO	IPPARM_ VENDOR_ PRODUCT_ID	Any state after Offered or Proceding.	char*
Vendor Version ID	IPSET_ VENDORINFO	IPPARM_ VENDOR_ VERSION_ID	Any state after Offered or Proceding.	char*
H.221 Nonstandard Information	IPSET_ VENDORINFO	IPPARM_ H221NONSTD	Any state after Offered or Proceding.	IP_H221_ NON STANDARD*
Conference ID	IPSET_ CONFERENCE	IPPARM_ CONFERENCE_ ID	Any state after Offered or Proceding.	Uint[8]
Conference Goal	IPSET_ CONFERENCE	IPPARM_ CONFERENCE_ GOAL	Any state after Offered or Proceding.	char*

Table 8. Retrievable Call Information

Parameter	Set ID	Parameter ID	When Information Can Be Retrieved	Datatype in value_buf Field (See Note Below)

Note:

1. This field is the value_buf field in the GC_PARM_DATA structure associated with the GCEV_EXTENSION event generated in response to the **gc_Extension( )** function requesting the information.

2. Display information, user to user information, phone list, nonstandard object ID, nonstandard data, vendor information, and H221 nonstandard information may not be present.

3.Vendor information is included in a Q931 SETUP message received from a peer. The vendor information that is sent with a Q931 SETUP message to a peer when issuing outbound calls is set up in the *config.val* file.

4. The nonstandard object id and nonstandard data parameters described here refer to nonstandard data contained in a SETUP message for example. This should not be confused with the nonstandard data included in protocol messages sent using **gc_Extension( )** which can be retrieved from the metaevent associated with a GCEV_EXTENSION event.

If an attempt is made to retrieve information in a state in which the information is not available, no error is generated. The GC_PARM_BLK associated with the generated GCEV_EXTENSION event will not contain the requested information. If phone list and display information are requested and only phone list is available, then only phone list information is available in the GC_PARM_BLK. An error is generated if there is an internal error (such as memory cannot be allocated).

All call information is available until a **gc_ReleaseCall( )** is issued. Coder information is available after a disconnect.

4.3.1. Retrieving Nonstandard Data From Protocol Messages

The following messages use nonstandard data:

- Setup (H.245)
- Facility (Q.931)
- H.245 message with nonstandard data (H.245)

Table 9 indicates when nonstandard information (object ID and data) should be retrieved for each type of message.

Table 9. Retrieving Nonstandard Data and ID from Messages

Message Elements	Message Type	When Information can be Retrieved	How to Retrieve the Information
Nonstandard Object ID and Data	Setup message (H.245)	When GCEV_OFFERED is received.	Use **gc_Extension( )** with the GETINFO extension ID.
Nonstandard Object ID and Data	Facility (Q.931)	When GCEV_EXTENSION event is received.	Retrieve the message information from the metaevent associated with the GCEV_EXTENSION event.
Nonstandard Object ID and Data	H.245 message with nonstandard data (H.245)	When GCEV_EXTENSION event is received.	Retrieve the message information from the metaevent associated with the GCEV_EXTENSION event.

NOTE: The nonstandard object ID and data associated with a Setup message should be retrieved immediately after the corresponding event since there is the possibility of the arrival of a facility message that could override the nonstandard object ID and data values.

4.4. Sending Protocol Messages

The following message types are supported:

- UII Message (H.245)
- Nonstandard Command Message (H.245)
- Facility Messages (Q.931)

Table 10 summarizes the set IDs and parameter IDs used to send the messages and describes the call states in which each message should be sent.

Table 10. Summary of Protocol Messages that Can be Sent

Type	Set ID	Parameter ID	When Message Should be Sent
UII message (H.245)	IPSET_ USERINPUT INDICATION	IPPARM_UII_ ALPHANUMERIC	When the call is in the Connected state only.
Nonstandard Command Message (H.245)	IPSET_MSG_H245	IPPARM_MSGTYPE (set to IP_MSGTYPE_H245_ COMMAND	When the call is in the Connected state only.
Facility message (Q.931)	IPSET_MSG_Q931	IPPARM_MSGTYPE (set to IP_MSGTYPE_Q931_ FACILITY	In any call state.

NOTES: 1. Nonstandard UII messages are not supported.

2. When using the **gc_Extension( )** function with the IPEXTID_SENDMSG extension ID, the **gc_Extension( )** function must be called in synchronous mode, that is, the **mode** parameter must be set to EV_SYNC.

4.4.1. UII Message (H.245)

Use the **gc_Extension( )** function in synchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG to send UII messages. At the receiving end, a GCEV_EXTENSION event with the same **ext_id** value is generated. The metaevent associated with the GCEV_EXTENSION event will contain all of the data in the message.

The relevant parameter set IDs and parameter IDs for this purpose are:

- IPSET_MSG_H245
  - IPPARM_MSGTYPE - Set to IP_MSGTYPE_H245_INDICATION.
- IPSET_USERINPUTINDICATION
  - IPPARM_UII_ALPHANUMERIC

NOTES: 1. The message type (IPPARM_MSGTYPE) is mandatory.

2. UII messages are only sent out to the IP network if the **PrmDTMFXferMode** parameter in the *.config* file is set to **1** (out-of-band DTMF transfer disabled). See Section 4.5, "Configuring DTMF Handling", on page 58 for more information.
3. The **gc_Extension( )** function must be called in synchronous mode, that is, the **mode** parameter must be set to EV_SYNC.

See Section 6.4, "IPSET_MSG_H245 Parameter Set", on page 74 and Section 6.7, "IPSET_USERINPUTINDICATION Parameter Set", on page 76 for more information.

The following code an example of how to send a UII message:

```
.
.
.
/* H245 UII Message - Note: Only works if PrmDtmfXferMode set to 1 in fcd file.*/

rc = gc_util_insert_parm_val(&t_PrmBlkp, IPSET_MSG_H245, IPPARM_MSGTYPE,
     sizeof(int), IP_MSGTYPE_H245_INDICATION);
rc = gc_util_insert_parm_ref(&t_PrmBlkp, IPSET_USERINPUTINDICATION,
     IPPARM_UII_ALPHANUMERIC, 6 "12345");

if (rc == -1) {
    printf("Fail to insert parm");
    return -1;}
else
    printf("Sending IP H245 UII Message");

gc_Extension(GCTGT_GCLIB_CRN, crn, IPEXTID_SENDMSG, t_PrmBlkp, &t_RetBlkp,
             EV_SYNC);
gc_util_delete_parm(t_PrmBlkp);
.
.
.
```

4.4.2. Nonstandard Command Message (H.245)

Use the **gc_Extension( )** function in synchronous mode with an **ext_ id** (extension ID) of IPEXTID_SENDMSG to send nonstandard command messages. At the receiving end, a GCEV_EXTENSION event with the same **ext_id** value is generated. The metaevent associated with the GCEV_EXTENSION event will contain all of the data in the message.

The relevant parameter set IDs and parameter IDs for this purpose are:

- IPSET_MSG_H245
  - IPPARM_MSGTYPE - Set to IP_MSGTYPE_H245_COMMAND.
- IPSET_NONSTANDARDDATA
  - IPPARM_NONSTANDARDDATA_OBJID - Object ID string.
  - IPPARM_NONSTANDARDDATA_DATA - Actual nonstandard data.

NOTES: 1. The message type (IPPARM_MSGTYPE) is mandatory. At least one other information element must be included.

2. The **gc_Extension( )** function must be called in synchronous mode, that is, the **mode** parameter must be set to EV_SYNC.

See Section 6.5, "IPSET_MSG_Q931 Parameter Set", on page 75 and Section 6.6, "IPSET_NONSTANDARDDATA Parameter Set", on page 75 for more information.

```
.
.
.
/* H245 Command with ObjId and data */

rc = gc_util_insert_parm_val(&t_PrmBlkp, IPSET_MSG_H245, IPPARM_MSGTYPE,
                             sizeof(int), IP_MSGTYPE_H245_COMMAND);

rc = gc_util_insert_parm_ref(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                             IPPARM_NONSTANDARDDATA_OBJID, ObjLen+1, ObjId);

rc = gc_util_insert_parm_ref(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                     IPPARM_NONSTANDARDDATA_DATA, DataLen+1, data + ObjLen + 1);

if (rc == -1) {
    printf("Fail to insert parm");
    return -1; }
else
    printf("Sending IP H245 Command Message");

gc_Extension(GCTGT_GCLIB_CRN, crn, IPEXTID_SENDMSG, t_PrmBlkp, &t_RetBlkp,
             EV_SYNC);
gc_util_delete_parm(t_PrmBlkp);
.
.
.
```

4.4.3. Facility Message (Q.931)

Use the **gc_Extension( )** function in synchronous mode with an **ext_ id** (extension ID) of IPEXTID_SENDMSG to send facility (Q.931 Facility) messages. At the receiving end, a GCEV_EXTENSION event with the same **ext_id** value is generated. The metaevent associated with the GCEV_EXTENSION event will contain all of the data in the message.

The relevant parameter set IDs and parameter IDs are:

- IPSET_MSG_Q931
  - IPPARM_MSGTYPE - Set to IP_MSGTYPE_Q931_FACILITY.
- IPSET_NONSTANDARDDATA
  - IPPARM_NONSTANDARDDATA_OBJID - Object ID string.
  - IPPARM_NONSTANDARDDATA_DATA - Actual nonstandard data.

NOTES: **1.** The message type (IPPARM_MSGTYPE) is mandatory. At least one other information element must be included.

2. The **gc_Extension( )** function must be called in synchronous mode, that is, the **mode** parameter must be set to EV_SYNC.

See Section 6.5, “IPSET_MSG_Q931 Parameter Set”, on page 75 and Section 6.6, “IPSET_NONSTANDARDDATA Parameter Set”, on page 75 for more information.

The following code shows how to set up and send a Q.931 facility message.

```
.
.
/* Q931 Facility Message */

rc = gc_util_insert_parm_val(&t_PrmBlkp, IPSET_MSG_Q931, IPPARM_MSGTYPE,
                             sizeof(int), IP_MSGTYPE_Q931_FACILITY);

rc = gc_util_insert_parm_ref(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                      IPPARM_NONSTANDARDDATA_OBJID, ObjLen+1, ObjId);

rc = gc_util_insert_parm_ref(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                      IPPARM_NONSTANDARDDATA_DATA, DataLen+1, data + ObjLen + 1);

if (rc == -1) {
    printf("Fail to insert parm");
    return -1;}
```

```
else
    printf("Sending IP Q931 Facility Message");

gc_Extension(GCTGT_GCLIB_CRN, crn, IPEXTID_SENDMSG, t_PrmBlkp, &t_RetBlkp,
             EV_SYNC);
gc_util_delete_parm(t_PrmBlkp);
.
.
```

4.4.4. Sending Facility or UII Message Scenario

The **gc_Extension( )** function can be used to send Q.931 facility messages or Q.245 UII messages. Figure 8 shows this scenario.

A Q.245 UII message can only be sent when a call is in the connected state. A Q.931 facility message can be sent in any call state.

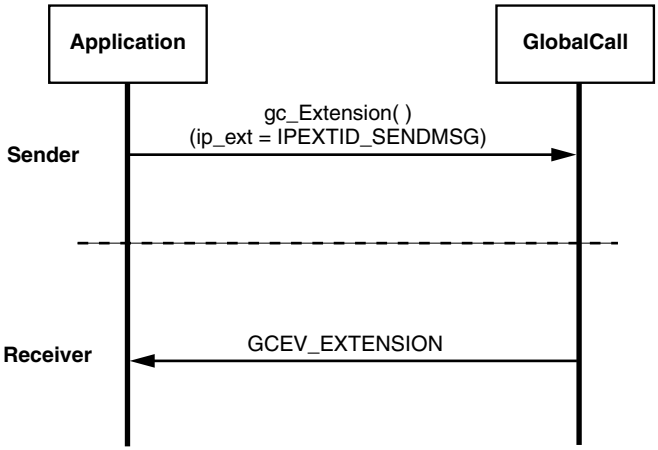


Figure 8. Sending Protocol Messages

4.5. Configuring DTMF Handling

When using Global Call with a DM3 board that uses an embedded stack, by default, DTMF digits are automatically transferred from the IP network to the CT Bus and

vice versa. However, the behavior can be configured by setting the value of the **PrmDTMFXferMode** parameter in the *.config* file.

When the **PrmDTMFXferMode** parameter is set to a value of **2** (default), out-of-band (OOB) DTMF transfer is **enabled** and the following behavior applies:

- DTMF digits from the CT Bus are sent out-of-band to the IP network.
- H.245 UII messages are used to generate DTMF digits which are sent to the CT Bus.
- In-band DTMF digits are set to the CT Bus. In this case, audio from the IP network is transmitted to the CT Bus and any DTMF digits are also sent. If the quality is not adequate, use out-of-band (OOB) DTMF transfer.
- H.245 UII messages from the application (**gc_Extension( )** with an extension ID of IPEXTID_SENDMSG) are ignored.

When the **PrmDTMFXferMode** parameter is set to a value of **1**, out-of-band (OOB) DTMF transfer is **disabled** and the following behavior applies:

- DTMF digits from the CT Bus are sent inband to the IP network.
- H.245 UII messages from the peer (containing out-of-band DTMF digits) are sent to the application.
- In-band DTMF digits are set to the CT Bus.
- H.245 UII messages from the application (**gc_Extension( )** with an extension ID of IPEXTID_SENDMSG) are sent to the IP network.

The default value of the **PrmDTMFXferMode** parameter (2) is appropriate in most cases. However, in a terminate configuration, when it is necessary to send DTMF digits to the CT Bus and a voice channel (dxxxBwCz) is not available to play to the CT Bus, **PrmDTMFXferMode** can be set to 1.

NOTE: When out-of-band (OOB) DTMF transfer is disabled (**PrmDTMFXferMode** set to 1), the application can only send OOB DTMF digits (H245 UII messages) to the IP network. It cannot send DTMF digits to the CT Bus or inband DTMF digits to the IP network. Similarly, the application can only detect OOB DTMF digits, it cannot detect inband DTMF digits or DTMF digits from the CT Bus.

Two other possible values of the **PrmDTMFXFerMode** parameter are 0 and 3. A value of 0 disables DTMF transmission, while the value 3 is

reserved for future use. See the "DTMF Parameters" section in the *DM3 IPLink User's Guide* for more information.

4.6. Call Routing

When using Global Call with a DM3 board that uses an embedded stack, routing is not required to associate an IP network device with an IP Media device, since each IP Media device is preallocated (bound) to a specific network device.

The **gc_GetXmitSlot( )** and **gc_Listen( )** functions can be used for call routing where an IP Media device can be routed to a non-IP voice or network resource.

The following pseudo code indicates how to perform full duplex routing between a non-IP network device (dtiB1T1) and an IP device (ipm_B1C1):

```
/* open dtiB1T1 */
gc_OpenEx(&dtidev, ":N_dtiB1T1:V_dxxxB1C1:P_pdk_ar_r2.io");
/* the :V part is optional and the :P part could specify another protocol /*

/* open ipmB1C1 */
gc_OpenEx(&ipdev, ":N_iptB1T1:M_ipmB1C1:P_H323_R");

/* get transmit time slot info of dtiB1T1 */
gc_GetXmitSlot(dtidev, &dti_sctsinfo);

/* make ipmB1C1 listen to transmit time slot of dtiB1T1 */
gc_Listen(ipdev, &dti_sctsinfo);

/* get transmit time slot info of ipmB1C1 */
gc_GetXmitSlot(ipdev, &ipm_sctsinfo);

/* make dtiB1T1 listen to transmit time slot of ipmB1C1 */
gc_Listen(dtidev, &ipm_sctsinfo);
```

4.7. Setting and Retrieving Quality of Service (QoS) Thresholds

When using Global Call with other technologies (such as E-1 CAS, T-1 Robbed Bit etc.), alarms are managed and reported on the network device. For example, when **gc_OpenEx( )** is issued, specifying both a network device (dtiB1T1) and a voice device (dxxxB1C1) in the **devicename** parameter, the function retrieves a Global Call line device. This Global Call line device can be used directly in Global Call

Alarm Management System (GCAMS) functions to manage alarms on the network device.

When using Global Call with IP technology, alarms such as QoS alarms, are more directly related to the media processing and are therefore reported on the media device not on the network device. When **gc_OpenEx( )** is issued, specifying both a network device (iptB1T1) and a media device (ipmB1C1) in the **devicename** parameter, two Global Call line devices are created:

- The first Global Call line device corresponds to the network device and is retrieved in the **gc_OpenEx( )** function.
- The second Global Call line device corresponds to the media device and is retrieved using the **gc_GetResourceH( )** function. This is the line device that must be used with GCAMS functions to manage QoS alarms. See the *Global Call Application Developer's Guide* for more information about GCAMS.

NOTE: Applications **must** include the *ipmlib.h* and *gcipmlib.h* header files before Global Call can be used to set or retrieve QoS threshold values.

4.7.1. Retrieving the Media Device Handle

To retrieve the Global Call line device corresponding to the media device, use the **gc_GetResourceH( )** function. See Section 3.1.11, "gc_GetResourceH( )", on page 25 for more information.

The Global Call line device corresponding to the media device is the device that must be used with GCAMS functions to manage QoS alarms.

4.7.2. Setting QoS Threshold Values

To set QoS threshold values, use the **gc_SetAlarmParm( )** function. See Section 3.1.17, "gc_SetAlarmParm( )", on page 33 for more information.

4.7.3. Retrieving QoS Threshold Values

To retrieve QoS threshold values, use the **gc_GetAlarmParm( )** function. See for Section 3.1.7, "gc_GetAlarmParm( )", on page 23 more information.

4.7.4. Handling QoS Alarms

When a GCEV_ALARM event occurs, use the Global Call Alarm Management System (GCAMS) functions (such as, **gc_AlarmNumber( )**) to retrieve information about the alarm that occurred. See the *Global Call Application Development Guide* for more information about the operation of GCAMS and the *Global Call API Software Reference* for more information about GCAMS functions.

4.7.5. Call Scenario Using QoS Threshold Values

The following scenario shows the sequence of calls required when setting or retrieving QoS threshold values:

1. Use the **gc_OpenEx( )** function to open the media device.
2. Use the **gc_GetResourceH( )** function to retrieve the media handle.
3. Use the **gc_SetAlarmNotifyAll( )** function with a value of ALARM_NOTIFY to enable the notification of alarms, that is, all GCEV_ALARM events.
4. Use the **gc_GetAlarmParm( )** function to get QoS thresholds.
5. Use the **gc_SetAlarmParm( )** function to set QoS thresholds.
6. When a GCEV_ALARM event occurs, use the Global Call Alarm Management System (GCAMS) functions such as, **gc_AlarmNumber( )**, to retrieve information about the alarm that occurred.

4.7.6. Sample Code for Managing QoS Threshold Alarms

The following code demonstrates how to set and retrieve QoS threshold values and how to handle alarm events related to QoS.

```
/*******************************************************************************
Routine:                QoS_sample_code
Assumptions/Warnings:   None
Description:            Illustrates the setting/getting of QoS thresholds
Parameters:             None
Returns:                None
*******************************************************************************/

void QoS_sample_code(void)
{
    LINEDEV                 ldev;
    int                     mediah;
```

```
    /* open IP device with media */
    if (gc_OpenEx(&ldev, ":N_iptB1T1:P_H323_NTSC:M_ipmB1C1", EV_SYNC, 0)
                != GC_SUCCESS)
    {
        /* handle gc_OpenEx() failure */
        return;
    }

    /* wait for GCEV_UNBLOCKED event (code not shown) */

    /* get device handle for media resource */
    if (gc_GetResourceH(ldev, &mediah, GC_MEDIADEVICE) != GC_SUCCESS)
    {
        /* handle gc_GetResourceH() failure */
        return;
    }

    /* enable alarm notification on media device (to get GCEV_ALARM event) */
    if (gc_SetAlarmNotifyAll(mediah, ALARM_SOURCE_ID_NETWORK_ID, ALARM_NOTIFY)
                              != GC_SUCCESS)
    {
        /* handle gc_SetAlarmNotifyAll() failure */
        return;
    }

    /* get default QoS threshold values */
    GetAlarmParm(mediah, ALARM_SOURCE_ID_NETWORK_ID,
                 ParmSetID_qosthreshold_alarm, EV_SYNC);
    /* change QoS thresholds */
    SetAlarmParm(mediah, ALARM_SOURCE_ID_NETWORK_ID,
                 ParmSetID_qosthreshold_alarm, EV_SYNC);

    /* get new QoS thresholds */
    GetAlarmParm(mediah, ALARM_SOURCE_ID_NETWORK_ID,
                 ParmSetID_qosthreshold_alarm, EV_SYNC);

    /* set up call and monitor for GCEV_ALARM events on media devices
            (code not shown)
                    .
                    .
                    .

    */

    /* close device */
    if (gc_Close(ldev) != GC_SUCCESS)
    {
        /* handle gc_Close() failure */
        return;
    }
}
```

```
/*******************************************************************************
Routine:                GetAlarmParm
Assumptions/Warnings:   None
Description:            calls gc_GetAlarmParm()
Parameters:             parameters to gc_GetAlarmParm() function call
Returns:                None
*******************************************************************************/

void GetAlarmParm(LINEDEV ldev, unsigned long aso_id, int ParmSetID,
                        unsigned long mode)
{
    ALARM_PARM_LIST alarm_parm_list;
    IPM_QOS_THRESHOLD_DATA QoS_parm[3];
    IPM_QOS_THRESHOLD_DATA *QoS_parmp;
    int n;

    /* get QoS thresholds for DTMFDISCARDED, LOSTPACKETS and JITTER */
    alarm_parm_list.n_parms = 3;
    for (n=0; n < alarm_parm_list.n_parms; n++)
    {
        alarm_parm_list.alarm_parm_fields[n].alarm_parm_data.pstruct
            = (void *) &QoS_parm[n];
    }

    QoS_parm[0].eQoSType = QOSTYPE_DTMFDISCARDED;
    QoS_parm[1].eQoSType = QOSTYPE_LOSTPACKETS;
    QoS_parm[2].eQoSType = QOSTYPE_JITTER;

    if (gc_GetAlarmParm(ldev, aso_id, ParmSetID, &alarm_parm_list, mode)
                        != GC_SUCCESS)
    {
        /* handle gc_GetAlarmParm() failure */
        return;
    }

    /* display threshold values retrieved */
    printf("n_parms = %d\n", alarm_parm_list.n_parms);

    for (n=0; n < alarm_parm_list.n_parms; n++)
    {
        QoS_parmp = alarm_parm_list.alarm_parm_fields[n].alarm_parm_data.pstruct;

        printf("QoS type = %d\n", QoS_parmp->eQoSType);
        printf("\tTime Interval = %u\n", QoS_parmp->unTimeInterval);
        printf("\tDebounce On = %u\n", QoS_parmp->unDebounceOn);
        printf("\tDebounce Off = %u\n", QoS_parmp->unDebounceOff);
        printf("\tFault Threshold = %u\n", QoS_parmp->unFaultThreshold);
        printf("\tPercent Success Threshold = %u\n",
                        QoS_parmp->unPercentSuccessThreshold);
        printf("\tPercent Fail Threshold = %u\n",
                        QoS_parmp->unPercentFailThreshold);
```

```
        printf("\n\n");
    }
}

/*****************************************************************************
Routine:                SetAlarmParm
Assumptions/Warnings:   None.
Description:            calls gc_SetAlarmParm()
Parameters:             parameters to gc_SetAlarmParm() function call
Returns:                None
*****************************************************************************/

void SetAlarmParm(LINEDEV ldev, unsigned long aso_id, int ParmSetID,
                        unsigned long mode)
{
    ALARM_PARM_LIST alarm_parm_list;
    IPM_QOS_THRESHOLD_DATA QoS_parm[1];

    alarm_parm_list.n_parms = 1;
    alarm_parm_list.alarm_parm_fields[0].alarm_parm_data.pstruct =
        (void *) &QoS_parm[0];

    QoS_parm[0].eQoSType = QOSTYPE_LOSTPACKETS;
    QoS_parm[0].unTimeInterval = 50;
    QoS_parm[0].unDebounceOn = 100;
    QoS_parm[0].unDebounceOff = 100;
    QoS_parm[0].unFaultThreshold = 10;
    QoS_parm[0].unPercentSuccessThreshold = 90;
    QoS_parm[0].unPercentFailThreshold = 10;

    if (gc_SetAlarmParm(ldev, aso_id, ParmSetID, &alarm_parm_list, mode)
                        != GC_SUCCESS)
    {
        /* handle gc_SetAlarmParm() failure */
        return;
    }
}

/*****************************************************************************
Routine:                handle_alarm
Assumptions/Warnings:   None.
Description:            handles the GCEV_ALARM event
Parameters:             metaevent:  pointer to METAEVENT for event
Returns:                None
*****************************************************************************/

void handle_alarm(METAEVENT *metaevent)
{
    ALARM_LIST      alarm_list;
    int             i;
    unsigned long   aso_id;
    long            alarm_number;
```

```
char            *alarm_name;

/*
 * retrieve and display aso id, alarm number, alarm name
 * associated with GCEV_ALARM that occurred
 */
if (gc_AlarmSourceObjectID(metaevent, &aso_id) != GC_SUCCESS)
{
    /* handle gc_AlarmSourceObjectID() failure */
    return;
}
printf("aso id = 0x%ul\n", aso_id);

if (gc_AlarmNumber(metaevent, &alarm_number) != GC_SUCCESS)
{
    /* handle gc_AlarmNumber() failure */
    return;
}
printf("alarm number = 0x%lx\n", alarm_number);

if (gc_AlarmNumberToName(aso_id, alarm_number, &alarm_name) != GC_SUCCESS)
{
    /* handle gc_AlarmNumberToName() failure */
    return;
}
printf ("alarm name = %s\n", alarm_name);

/* retrieve and display ON/OFF status of all alarms */
if (gc_GetAlarmConfiguration(metaevent->linedev, ALARM_SOURCE_ID_NETWORK_ID,
        &alarm_list, ALARM_CONFIG_STATUS) != GC_SUCCESS)
{
    /* handle gc_GetAlarmConfiguration() failure */
    return;
}
for (i = 0; i< alarm_list.n_alarms; i++)
{
    printf("alarm number = %d",alarm_list.alarm_fields[i].alarm_number);

    switch (alarm_list.alarm_fields[i].alarm_data.intvalue)
    {
    case ALARM_ON:
        printf("\talarm status = ALARM_ON\n");
        break;
    case ALARM_OFF:
        printf("\talarm status = ALARM_OFF\n");
        break;
    default:
        printf("\talarm status = %d (unknown)\n",
```

```
                alarm_list.alarm_fields[i].alarm_data.intvalue);
            break;
        }
    }
}
```

5. Building Applications

5.1. Required System Software

The System Software must be installed on the development system.

For Windows systems, make sure that you select the Custom Install option and select the DM3, IPLink, and Global Call options when installing the System Software.

For Linux systems, make sure that the following packages are installed:

- DLGCcom
- DLGCdmdev
- DLGCiplnk
- DLGCgc

This ensures that all the required libraries are installed in the correct directories.

5.2. Global Call IP-Specific Header Files

In addition to the generic header files required by Global Call, the following technology-specific header files are also required when building IP applications that use Global Call for call control:

- *gcip.h*
- *gcip_defs.h*
- *ipmlib.h* - for Quality of Service (QoS) features
- *gcipmlib.h* - for Quality of Service (QoS) features

5.3. Pre-Call Setup

Before opening an IP line device using **gc_OpenEx( )**, the IP address to be used for the device must be configured.

In Windows systems, the IP address is configured using the Dialogic Configuration Manager (DCM) when downloading the firmware to the board.

5.4. Call Setup and Teardown

An application may use the slow connect scenario or the fast connect scenario for a call. See Section 4.1, "Using Fast and Slow Start Connections", on page 41 for more information.

6. IP-Specific Parameter Set Reference

The parameter set IDs and parameter IDs described in this chapter are defined in the *gcip.h* header file. Table 11 summarizes the parameter set IDs and parameter IDs used by Global Call in an IP environment.

The meaning of the columns in the table following are:

- **Set** - Indicates the Global Call functions used to set the parameter information.
- **Send** - Indicates the Global Call functions used to send the information to a peer end point.
- **Retrieve** - Indicates the Global Call function used to retrieve information that was sent by a peer end point.

Table 11. Summary of Parameter IDs and Set ID

Parameter ID	Set ID	Set	Send	Retrieve
IPPARM_ RTCPINFO	IPSET CALLINFO	---	---	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ LOCAL_ CAPABILITY	GCSET_ CHAN_ CAPABILITY	gc_SetConfigData( ) gc_SetUserInfo( ) (GC_ALLCALLS) gc_SetUserInfo( ) (GC_SINGLECALL)	gc_AnswerCall( ) and gc_MakeCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ DISPLAY	IPSET_ CALLINFO	gc_SetUserInfo( ) (GC_SINGLECALL) gc_MakeCall( )	gc_AnswerCall( ) and gc_MakeCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ USERUSER_ INFO	IPSET_ CALLINFO	gc_SetUserInfo( ) (GC_SINGLECALL) gc_MakeCall( )	gc_MakeCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ CALL DURATION	IPSET_ CALLINFO	---	---	gc_Extension( ) (IPEXTID_ GETINFO)

Table 11. Summary of Parameter IDs and Set ID

Parameter ID	Set ID	Set	Send	Retrieve
IPPARM_ PHONELIST	IPSET_ CALLINFO	gc_SetUserInfo( ) (GC_SINGLECALL) gc_MakeCall( )	gc_MakeCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ NON STANDARD DATA_OBJID	IPSET_ CALLINFO	gc_SetUserInfo( ) (GC_SINGLECALL) gc_MakeCall( )	gc_AnswerCall( ), gc_MakeCall( ) and gc_DropCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ NON STANDARD DATA_DATA	IPSET_ CALLINFO	gc_SetUserInfo( ) (GC_SINGLECALL) gc_MakeCall( )	gc_AnswerCall( ), gc_MakeCall( ) and gc_DropCall( )	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ VENDOR_ PRODUCT_ID	IPSET_ VENDOR INFO	---	---	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ VENDOR_ VERSION_ID	IPSET_ VENDOR INFO	---	---	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ H221NONSTD	IPSET_ VENDOR INFO	---	---	gc_Extension( ) (IPEXTID_ GETINFO)
IPPARM_ MSGTYPE	IPSET_ MSG_H245	---	gc_Extension( ) (IPEXTID_ SENDMSG)	gc_Extension( ) (IPEXTID_ SENDMSG)
IPPARM_ MSGTYPE	IPSET_ MSG_Q931	---	gc_Extension( ) (IPEXTID_ SENDMSG)	gc_Extension( ) (IPEXTID_ SENDMSG)
IPPARM_ UII_ ALPHA NUMERIC	IPSET_ USERINPUT INDICATION	---	gc_Extension( ) (IPEXTID_ SENDMSG)	gc_Extension( ) (IPEXTID_ SENDMSG)
IPPARM_ CONFERENCE_ GOAL	IPSET_CONF ERENCE	---	---	gc_Extension( ) (IPEXTID_ GETINFO)

Table 11. Summary of Parameter IDs and Set ID

Parameter ID	Set ID	Set	Send	Retrieve
IPPARM_ CONFERENCE_ ID	IPSET_CONF ERENCE	---	---	gc_Extension( ) (IPEXTID_ GETINFO)

6.1. GC_SETCHAN_CAPABILITY Parameter Set

Table 12 shows the parameter IDs in the GC_SETCHAN_CAPABILITY parameter set.

Table 12. GC_SETCHAN_CAPABILITY Parameter Set

Parameter ID	Description	Type
IPPARM_LOCAL_ CAPABILITY	Used to pass the local capabilities to be associated with a line device, or a subset thereof when used to communicate selected capabilities for a call.	IP_CAPABILITY

6.2. IPSET_CALLINFO Parameter Set

Table 13 shows the parameter IDs in the IPSET_CALLINFO parameter set.

Table 13. IPSET_CALLINFO Parameter Set

Parameter ID	Type	Description
IPPARM_CALLDURATION	String	The duration of a call. This information is available after a call is dropped (**gc_DropCall( )** but before the call is released (**gc_ReleaseCall( )**).
IPPARM_DISPLAY	Uint8[ ]	Display information. This information can be used by a peer as additional address information. The maximum size is 82.

Table 13. IPSET_CALLINFO Parameter Set

Parameter ID	Type	Description
IPPARM_PHONELIST	String	Phone number to dial at remote end point. **Note**: When issuing a **gc_MakeCall( )**, this information can also be sent through the **numberstr** parameter. See Section 3.1.14, "gc_MakeCall( )", on page 26 for more information.
IPPARM_RTCPINFO	IP_RTCPINFO	Updated RTCP information
IPPARM_USERUSER_INFO	Uint8[ ]	User-to-user information. The maximum size is 131.
Note: For parameter IDs of type String, the length of the string when used in a GC_PARM_BLK is the length of the string plus 1.		

6.3. IPSET_CONFERENCE Parameter Set

Table 14 shows the parameter IDs in the IPSET_CONFERENCE parameter set.

Table 14. IPSET_CONFERENCE Parameter Set

Parameter ID	Type	Description
IPPARM_CONFERENCE_ID	Uint[8]	The conference identifier.
IPPARM_CONFERENCE_GOAL	String	The conference functionality to be achieved.
Note: For parameter IDs of type String, the length of the string when used in a GC_PARM_BLK is the length of the string plus 1.		

6.4. IPSET_MSG_H245 Parameter Set

Table 15 shows the parameter IDs in the IPSET_MSG_H245 parameter set.

This parameter set is used with the **gc_Extension( )** and the IPEXTID_SENDMSG extension and encapsulates all the parameters required to send an H.245 message.

Table 15. IPSET_MSG_H245 Parameter Set

Parameter IDs	Type	Description
IPPARM_MSGTYPE	int	Possible values for H.245 messages are: • IP_MSGTYPE_H245_COMMAND • IP_MSGTYPE_H245_INDICATION

6.5. IPSET_MSG_Q931 Parameter Set

Table 16 shows the parameter IDs in the IPSET_MSG_Q931 parameter set.

This parameter set is used with the **gc_Extension( )** and the IPEXTID_SENDMSG extension and encapsulates all the parameters required to send an H.245 message.

Table 16. IPSET_MSG_Q931 Parameter Set

Parameter IDs	Type	Description
IPPARM_MSGTYPE	int	Possible values for Q.931 messages are: • IP_MSGTYPE_Q931_FACILITY

6.6. IPSET_NONSTANDARDDATA Parameter Set

Table 17 shows the parameter IDs in the IPSET_NONSTANDARDDATA parameter set.

Table 17. IPSET_NONSTANDARDDATA Parameter Set

Parameter IDs	Type	Description
IPPARM_NONSTANDARDDATA_DATA	String	Contains the non-standard data supplied, if any. If non-standard data was not supplied, this parameter should not be present in the parm block.

Table 17. IPSET_NONSTANDARDDATA Parameter Set

Parameter IDs	Type	Description
IPPARM_NONSTANDARDDATA_OBJID	String	Contains the non-standard object ID supplied, if any. If a non-standard object ID was not provided, this parameter should not be present in the parm block.
Note: For parameter IDs of type String, the length of the string when used in a GC_PARM_BLK is the length of the string plus 1.		

6.7. IPSET_USERINPUTINDICATION Parameter Set

Table 18 shows the parameter IDs in the IPSET_USERINPUTINDICATION parameter set.

Table 18. IPSET_USERINPUTINDICATION Parameter Set

Parameter IDs	Type	Description
IPPARM_UII_ALPHANUMERIC	String	Used to send user input indication (UII) messages. The **PrmDTMFXferMode** parameter in the *.config* file determines the behavior of this parameter ID that can be used to: • Send DTMF digits • Send message information that a peer can retrieve by examining the structure asssociated with the GCEV_EXTENSION event generated at the peer. • See Section 4.5, "Configuring DTMF Handling", on page 58 for more information.
Note: For parameter IDs of type String, the length of the string when used in a GC_PARM_BLK is the length of the string plus 1.		

6.8. IPSET_VENDORINFO Parameter Set

Table 19 shows the parameter IDs in the IPSET_VENDORINFO parameter set.

Table 19. IPSET_VENDORINFO Parameter Set

Parameter IDs	Type	Description
IPPARM_H221NONSTD	IP_H221NONSTANDARD	Contains country code, extension code and manufacturer code.
IPPARM_VENDOR_ PRODUCT_ID	String	Vendor product identifier.
IPPARM_VENDOR_ VERSION_ID	String	Vendor version identifier.
Note: For parameter IDs of type String, the length of the string when used in a GC_PARM_BLK is the length of the string plus 1.		

7. Data Structure Reference

This chapter describes data structures that are specific to IP technology. These data structures are defined in the *gcip.h* header file.

7.1. IP_AUDIO_CAPABILITY

The IP_AUDIO_CAPABILITY data structure is used to allow some minimum set of information to be exchanged together with the audio codec identifier. The data structure definition is as follows:

```
typedef struct
{
    unsigned long   frames_per_pkt;
    long            VAD;
} IP_AUDIO_CAPABILITY;
```

Table 20 describes the meaning of each field in the IP_AUDIO_CAPABILITY data structure.

Table 20. IP_AUDIO_CAPABILITY Field Descriptions

Field	Description
frames_per_pkt	When bundling more than one audio frame into a single transport packet, this value should represent the maximum number of frames per packet that will be sent on the wire. When set to zero, indicates that the exact number of frames per packet is not known, or that the data is not applicable. **Note**: For G.711 coders, this field represents the frame size (for example, 10 msec); the frames per packet value is fixed at 1 fpp. For other coders this field represents the frames per packet and the frame size is fixed. See Section 4.2.5, “Setting Coder Information”, on page 46 for more information.
VAD	Applies to audio algorithms that support the concept of voice activated detection (VAD) only. Possible values are: • GCPV_ENABLE - VAD enabled • GCPV_DISABLE - VAD disabled

7.2. IP_CAPABILITY

The IP_CAPABILITY data structure provides a level of capability information in addition to simply the capability or codec identifier.

NOTE: The IP_CAPABILITY data structure is not intended to provide all the flexibility of the H.245 terminal capability structure, but provides a first level of useful information in addition to the capability or codec identifier.

The data structure definition is as follows.

```
typedef struct
{
    int                     capability;
    int                     type;
    int                     direction;
    int                     payload_type;
    IP_CAPABILITY_UNION     extra;
    char                    rfu[0x10];
} IP_CAPABILITY;
```

Table 21 describes the meaning of each field in the IP_CAPABILITY data structure.

Table 21. IP_CAPABILITY Field Descriptions

Field	Description
capability	The IP Media capability for this structure. Possible values are: • GCCAP_AUDIO_G711Alaw64k • GCCAP_AUDIO_G711Ulaw64k • GCCAP_AUDIO_g7231_5_3k • GCCAP_AUDIO_g7231_6_3k • GCCAP_AUDIO_g729AnnexA • GCCAP_AUDIO_g729AnnexAwAnnexB • GCCAP_AUDIO_gsmFullRate
type	The category of capability specified in this structure. Indicates which member of the IP_CAPABILITY_UNION union is being used. Possible values are: • GCCAPTYPE_AUDIO - Audio

Table 21. IP_CAPABILITY Field Descriptions (Continued)

Field	Description
direction	The capability direction code for this capability. Possible values are: • IP_CAP_DIR_LCLTRANSMIT - Indicates a transmit capability for the local endpoint. • IP_CAP_DIR_LCLRECIEVE - Indicates a receive capability for the local endpoint
payload_type	The payload type. When using a standard payload type, set the value of this field to IP_USE_STANDARD_PAYLOADTYPE. When using a non-standard payload type, use this field to specify the RTP payload type that will be used in conjunction with the coder specified in the capability field in this structure.
extra	The contents of the IP_CAPABILITY_UNION will be indicated by the type field.
rfu	Reserved for future use. Must be set to zero when not used.

7.3. IP_CAPABILITY_UNION

The IP_CAPABILITY_UNION union enables different capability categories to define their own additional parameters or interest. The union definition is as follows:

```
typedef union
{
    IP_AUDIO_CAPABILITY         audio;
    IP_VIDEO_CAPABILITY         video;
    IP_DATA_CAPABILITY          data;
} IP_CAPABILITY_UNION;
```

Table 22 describes the meaning of each field in the IP_CAPABILITY_UNION union.

Table 22. IP_CAPABILITY_UNION Field Descriptions

Field	Description
audio	A structure that represents the audio capability. See Section 7.1, "IP_AUDIO_CAPABILITY", on page 79 for more information.
video	Not supported when using Global Call with a DM3 board that uses an embedded stack.
data	Not supported when using Global Call with a DM3 board that uses an embedded stack.

7.4. IP_H221NONSTANDARD

The IP_H221NONSTANDARD data structure is used to store H.221 non-standard data. The data structure definition is as follows:

```
typedef struct
{
    int   country_code;
    int   extension;
    int   manufacturer_code;
} IP_H221NONSTANDARD;
```

Table 23 describes the meaning of each field in the IP_H221NONSTANDARD data structure.

Table 23. IP_H221NONSTANDARD Field Descriptions

Field	Description
country_code	The country code.
extension	The extension number.
manufacturer_code	The manufacturer's code.

7.5. IP_MEDIA_STREAM_INFO

The IP_MEDIA_STREAM_INFO provides the RTP and RTCP port information along with information about the capability properties that follow. The data structure definition is as follows:

```
typedef struct
{
    char            RTPAddress[MAX_ADDR_LENGTH];
    int             RTPPort;
    char            RTCPAddress[MAX_ADDR_LENGTH];
    int             RTCPPort;
    char            rfu[3];
} IP_MEDIA_STREAM_INFO, *IP_MEDIA_STREAM_INFOP;
```

Table 24 describes the meaning of each field in the IP_MEDIA_STREAM_INFO data structure.

Table 24. IP_MEDIA_STREAM_INFO Field Descriptions

Field	Description
RTPAddress	The RTP IP address to which IP Media streaming takes place.
RTPPort	The RTP port on which IP Media streaming takes place.
RTCPAddress	For a receive channel, this is the IP address where the RTCP send reports are received, that is, the address to which the transmitter of the stream sends reports regarding the RTP packets being sent.
RTCPPort	The RTCP port on which streaming reports are sent and received.
rfu[3]	Reserved for future use.

7.6. IP_RTCPINFO

The IP_RTCPINFO data structure provides RTCP information. The data structure definition is as follows:

```
typedef struct
{
    RTCP_REPORT   local;
    RTCP_REPORT   remote;
} IP_RTCPINFO;
```

Table 25 describes the meaning of each field in the IP_RTCPINFO data structure.

Table 25. IP_RTCPINFO Field Descriptions

Field	Description
local	Contains RTCP information for the local end point. See Section 7.8, "RTCP_REPORT", on page 84 for more information.
remote	Contains RTCP information for the remote end point. See Section 7.8, "RTCP_REPORT", on page 84 for more information.

7.7. RTCP_RECEIVERREPORT

The RTCP_RECEIVERREPORT data structure provides RTCP sender information. The data structure definition is as follows:

```
typedef struct
{
    unsigned int   fraction_lost;
    unsigned int   cumulative_lost;
    unsigned int   sequence_number;
} RTCP_RECEIVERREPORT;
```

Table 26 describes the meaning of each field in the RTCP_RECEIVERREPORT data structure.

Table 26. RTCP_RECEIVERREPORT Field Descriptions

Field	Description
fraction_lost	Percentage of packets lost.
cumulative_lost	Number of packets lost.
sequence_number	Last sequence number received.

7.8. RTCP_REPORT

The RTCP_REPORT data structure provides RTCP sender and receiver information. The data structure definition is as follows:

```
typedef struct
{
    RTCP_SENDERREPORT     sender;
    RTCP_RECEIVERREPORT   receiver;
} RTCP_REPORT;
```

Table 27 describes the meaning of each field in the RTCP_REPORT data structure.

Table 27. RTCP_REPORT Field Descriptions

Field	Description
sender	Contains the sender information. See Section 7.9, "RTCP_SENDERREPORT", on page 85 for more information.
receiver	Contains the receiver information. See Section 7.7, "RTCP_RECEIVERREPORT", on page 84 for more information.

7.9. RTCP_SENDERREPORT

The RTCP_SENDERREPORT data structure provides RTCP sender information. The data structure definition is as follows:

```
typedef struct
{
    unsigned int   timestamp;
    unsigned int   tx_packets;
    unsigned int   tx_octets;
    unsigned int   send_indication;
} RTCP_SENDERREPORT;
```

Table 28 describes the meaning of each field in the RTCP_SENDERREPORT data structure.

Table 28. RTCP_SENDERREPORT Field Descriptions

Field	Description
timestamp	The time stamp of the RTCP packet transmission from the local or remote sender.
tx_packets	The number of packets sent by the local or remote sender.

Table 28. RTCP_SENDERREPORT Field Descriptions

Field	Description
tx_octets	The number of bytes sent by the local or remote sender.
send_indication	Indicates that the local or remote sender report has changed since the last report. Possible values are: • TRUE • FALSE

8. IP-Specific Event Cause Codes

The event cause codes described in this chapter are defined in the *gcip_defs.h* header file.

When a GCEV_DISCONNECTED event is received, use the **gc_ResultInfo( )** function to retrieve the reason or cause of that event.

Table 29 lists the supported IP-specific event cause codes and provides a description of each code. When using **gc_DropCall( )**, only event cause codes prefixed by IPEC_H2250 or IPEC_Q931 should be specified in the **cause** parameter. See Section 3.1.5, "gc_DropCall( )", on page 22 for more information.

Table 29. IP-Specific Event Cause Codes

Event Cause Code	Description
IPEC_H2250ReasonAdaptiveBusy	H2250 Reason.
IPEC_H2250ReasonBadFormatAddress	H2250 Reason.
IPEC_H2250ReasonCalledPartyNotRegistered	H2250 Reason.
IPEC_H2250ReasonCallerNotRegistered	H2250 Reason.
IPEC_H2250ReasonDestinationRejection	H2250 Reason.
IPEC_H2250ReasonFacilityCallDeflection	H2250 Reason.
IPEC_H2250ReasonGatekeeperResource	H2250 Reason.
IPEC_H2250ReasonGatewayResource	H2250 Reason.
IPEC_H2250ReasonInConf	H2250 Reason.
IPEC_H2250ReasonInvalidRevision	H2250 Reason.
IPEC_H2250ReasonMax	H2250 reason end marker.
IPEC_H2250ReasonMin = 1999	H2250 reason start marker.
IPEC_H2250ReasonNoBandwidth	H2250 Reason.
IPEC_H2250ReasonNoPermission	H2250 Reason.
IPEC_H2250ReasonSecurityDenied	H2250 Reason.

Table 29. IP-Specific Event Cause Codes (Continued)

Event Cause Code	Description
IPEC_H2250ReasonUndefinedReason	H2250 Reason.
IPEC_H2250ReasonUnreachableDestination	H2250 Reason.
IPEC_H2250ReasonUnreachableGatekeeper	H2250 Reason.
IPEC_Q931Cause01UnassignedNumber	Q.931 cause 01.
IPEC_Q931Cause02NoRouteToSpecifiedTransit Network	Q.931 cause 02.
IPEC_Q931Cause03NoRouteToDestination	Q.931 cause 03.
IPEC_Q931Cause06ChannelUnacceptable	Q.931 cause 06.
IPEC_Q931Cause07CallAwardedAndBeing DeliveredInAnEstablishedChannel	Q.931 cause 07.
IPEC_Q931Cause16NormalCallClearing	Q.931 cause 16.
IPEC_Q931Cause17UserBusy	Q.931 cause 17.
IPEC_Q931Cause18NoUserResponding	Q.931 cause 18.
IPEC_Q931Cause19UserAlertingNoAnswer	Q.931 cause 19.
IPEC_Q931Cause21CallRejected	Q.931 cause 21.
IPEC_Q931Cause22NumberChanged	Q.931 cause 22.
IPEC_Q931Cause26NonSelectUserClearing	Q.931 cause 26.
IPEC_Q931Cause27DestinationOutOfOrder	Q.931 cause 27.
IPEC_Q931Cause28InvalidNumberFormat IncompleteNumber	Q.931 cause 28.
IPEC_Q931Cause29FacilityRejected	Q.931 cause 29.
IPEC_Q931Cause30ResponseTo STATUSENQUIRY	Q.931 cause 30.
IPEC_Q931Cause31NormalUnspecified	Q.931 cause 31.
IPEC_Q931Cause34NoCircuitChannelAvailable	Q.931 cause 34.
IPEC_Q931Cause38NetworkOutOfOrder	Q.931 cause 38.
IPEC_Q931Cause41TemporaryFailure	Q.931 cause 41.

Table 29. IP-Specific Event Cause Codes (Continued)

Event Cause Code	Description
IPEC_Q931Cause42SwitchingEquipment Congestion	Q.931 cause 42.
IPEC_Q931Cause43AccessInformationDiscarded	Q.931 cause 43.
IPEC_Q931Cause44RequestedCircuitChannel NotAvailable	Q.931 cause 44.
IPEC_Q931Cause47ResourceUnavailable Unspecified	Q.931 cause 47.
IPEC_Q931Cause57BearerCapability NotAuthorized	Q.931 cause 57.
IPEC_Q931Cause58BearerCapability NotPresentlyAvailable	Q.931 cause 58.
IPEC_Q931Cause63ServiceOrOption NotAvailableUnspecified	Q.931 cause 63.
IPEC_Q931Cause65BearCapability NotImplemented	Q.931 cause 65.
IPEC_Q931Cause66ChannelType NotImplemented	Q.931 cause 66.
IPEC_Q931Cause69RequestedFacility NotImplemented	Q.931 cause 69.
IPEC_Q931Cause70OnlyRestrictedDigital InformationBearerCapabilityIsAvailable	Q.931 cause 70.
IPEC_Q931Cause79ServiceOrOption NotImplementedUnspecified	Q.931 cause 79.
IPEC_Q931Cause81InvalidCallReferenceValue	Q.931 cause 81.
IPEC_Q931Cause82IdentifiedChannel DoesNotExist	Q.931 cause 82.
IPEC_Q931Cause83AsuspendedCallExists ButThisCallIdentityDoesNot	Q.931 cause 83.
IPEC_Q931Cause84CallIdentityInUse	Q.931 cause 84.
IPEC_Q931Cause85NoCallSuspended	Q.931 cause 85.
IPEC_Q931Cause86CallHavingTheRequested CallIdentityHasBeenCleared	Q.931 cause 86.

Table 29. IP-Specific Event Cause Codes (Continued)

Event Cause Code	Description
IPEC_Q931Cause88IncompatibleDestination	Q.931 cause 88.
IPEC_Q931Cause91InvalidTransitNetwork Selection	Q.931 cause 91.
IPEC_Q931Cause95InvalidMessageUnspecified	Q.931 cause 95.
IPEC_Q931Cause96MandatoryInformation ElementMissing	Q.931 cause 96.
IPEC_Q931Cause97MessageTypeNonExistent OrNotImplemented	Q.931 cause 97.
IPEC_Q931Cause100InvalidInformationElement Contents	Q.931 cause 100.
IPEC_Q931Cause101MessageNotCompatible WithCallState	Q.931 cause 101.
IPEC_Q931Cause102RecoveryOnTimeExpiry	Q.931 cause 102.
IPEC_Q931Cause111ProtocolErrorUnspecified	Q.931 cause 111.
IPEC_Q931Cause127InterworkingUnspecified	Q.931 cause 127.
IPEC_Q931CauseMax	Q931 cause end marker.
IPEC_Q931CauseMin = 3000	Q931 cause start marker.

9. Called and Calling Party Address List Format

9.1. Called Party Address List

Called party address lists are formatted as shown as follows:

```
Called Party Address list ::= Called Party Address |
    Called Party Address Delimiter Party Address list

Called Party Address ::= Dialable Address | Name |
    E164ALIAS | Extension | Subaddress | Transport
    Address | Email Address | URL | Party Number |
    Transport Name
```

Where,

- Dialable Address ::= E164Address | E164Address ";" Dialable Address
- Name ::= "NAME:" H323ID
- E164ALIAS ::= "TEL:" E164Address
- Extension ::= "EXT:" E164Address | "EXTID : " H323ID
- Subaddress ::= "SUB:" E164Address
- Transport Address ::= "TA:" Transport Address Spec | "FTH : " Transport address Spec.
    - Transport Address Spec ::= Host Name":" Port Number | Host Name
        - Host Name ::= Host IP in decimal dotted notation.
- Email Address ::= "EMAIL :" email address
- URL Address ::= "URL : " URL
- PN Address ::= "PN :" party number ["$" party number type]
    - Party Number Type ::= (select either the numerical or string value from the following list):
        - **PUU cmPartyNumberPublicUnknown** - The numbering plan follows the E.163 and E.164Recommendations.

- **PUI cmPartyNumberPublicInternationalNumber** - The number digits carry a prefix indicating type of number according to national recommendations.
- **PUN cmPartyNumberPublicNationalNumber** - The number digits carry a prefix indicating the type of number according to national recommendations.
- **PUNS cmPartyNumberPublicNetworkSpecificNumber** - The number digits carry a prefix indicating the type of number according to network specifications.
- **PUS cmPartyNumberPublicSubscriberNumber** - Reserved for future use.
- **PUA cmPartyNumberPublicAbbreviatedNumber** - Valid only for the called party number at the outgoing access; the network substitutes appropriate number.
- **D cmPartyNumberDataPartyNumber** - Valid only for the called party number at the outgoing access; the network substitutes appropriate number.
- **T cmPartyNumberTelexPartyNumber** - Reserved for future use.
- **PRU cmPartyNumberPrivateUnknown** - Reserved for future use.
- **PRL2 cmPartyNumberPrivateLevel2RegionalNumber** - Level 2 regional subtype of private number.
- **PRL1 cmPartyNumberPrivateLevel1RegionalNumber** - Level 1 regional subtype of private number.
- **PRP cmPartyNumberPrivatePISNSpecificNumber** - PISN subtype of private number.
- **PRL cmPartyNumberPrivateLocalNumber** - Local subtype of private number.
- **PRA cmPartyNumberPrivateAbbreviatedNumber** - Abbreviated subtype of private number.
- **N cmPartyNumberNationalStandardPartyNumber)** - The number digits carry a prefix indicating standard type of number according to national recommendations.

- Transport Name ::= "TNAME :" Transport Address Spec

NOTES: 1. The delimiter is "," by default, but it may be changed using the **system.delimiter** stack configuration setting.

2. If the Dialable Address form of the address is used, it should be the last item in the list of address alternatives.

9.2. Calling Party Address List

Calling party address lists are formatted as follows:

```
Calling Party address list ::= Calling Party address |
    Calling Party address Delimiter |
    Calling Party address list

Calling Party address ::= Dialable Address | Name |
    E164ALIAS | Extension | Subaddress | Transport
    Address | Email Address | URL | Party Number |
    Transport Name
```

Where the format options Dialable Address, Name, etc. are as described in the Section 9.1, “Called Party Address List”, on page 91.

NOTE: If the Dialable Address form of the Party address is used, it should be the last item in the list of Party address alternatives.

9.3. Examples of Called and Calling Party Addresses

Some examples of called party and calling party addresses are:

- Called and Calling Party addresses: 1111;1111
- NAME: John, NAME: Jo
- TA:192.114.36.10

Glossary

alias: A nickname for a domain or host computer on the Internet.

codec: A device that converts analog voice signals to a digital form and vice versa. In this context, analog signals are converted into the payload of UDP packets for transmission over the internet. The codec also performs compression and decompression on a voice stream.

H.225.0: Specifies messages for call control including signaling, Registration Admission and Status (RAS), and the packetization and synchronization of media streams.

en-bloc mode: A mode where the setup message contains all the information required by the network to process the call, such as the called party address information.

H.245: H.245 is a standard that provides the call control mechanism that allows H.323-compatible terminals to connect to each other. H.245 provides a standard means for establishing audio and video connections. It specifies the signaling, flow control, and channeling for messages, requests, and commands. H.245 enables codec selection and capability negotiation within H.323. Bit rate, frame rate, picture format, and algorithm choices are some of the elements negotiated by H.245.

Gateway: Translates communication procedures and formats between networks, for example the interface between an IP network and the circuit-switched network (PSTN).

Gatekeeper: Manages a collection of H.323 entities (terminals, gateway, multipoint control units) in an H.323 zone.

H.323: H.323 is an ITU recommendation for a standard for interoperability in audio, video and data transmissions as well as Internet phone and voice-over-IP (VoIP). H.323 addresses call control and management for both point-to-point and multipoint conferences as well as gateway administration of IP Media traffic, bandwidth and user participation.

IP: Internet Protocol

IP Media Library: Used to control RTP streams.

Multipoint Control Unit (MCU): An endpoint that support conferences between three or more endpoints.

prefix: One or several digits dialed in front of a phone number, usually to indicate something to the phone system. For example, dialing a zero in front of a long distance number in the United States indicates to the phone company that you want operator assistance on a call.

Q.931: The Q.931 protocol defines how each H.323 layer interacts with peer layers, so that participants can interoperate with agreed upon formats. The Q.931 protocol resides within H.225.0. As part of H.323 call control, Q.931 is a link layer protocol for establishing connections and framing data.

RTP: Real-time Transport Protocol. Provides end-to-end network transport functions suitable for applications transmitting real-time data such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services.

RTCP: RTP Control Protocol (RTCP). Works in conjunction with RTP to allow the monitoring of data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTCP is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets.

silence suppression: See Voice Activation Detection (VAD).

VAD: Voice Activation Detection. In Voice over IP (VoIP), voice activation detection (VAD) is a technique that allows a data network carrying voice traffic over the Internet to detect the absence of audio and conserve bandwidth by preventing the transmission of "silent packets" over the network.

Index

H

I

L

M

N

P

Q

R

S

U

V