Global Call IP for Host Media Processing

Technology Guide

September 2004

05-2239-003

Publication Date: September 2004

Document Number: 05-2239-003

Intel Converged Communications, Inc.
1515 Route 10
Parsippany, NJ 07054

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

For **Products and Services Information**, visit the Intel Telecom Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Where to Buy Intel Telecom Products page at:
http://www.intel.com/buy/wtb/wtb1028.htm

Contents

Figures

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2039-003	September 2004	General Conditions for Call Transfers section (pg. 42): New section Using Fast Start and Slow Start Setup section (pg. 66): Added note about H.323 fast start when no coder is specified (PTR#33321) Summary of Call-Related Information that can be Set table (pg. 67): Added note that GC_SINGLECALL must be used for Call ID and SIP Message Information fields. Added entries for four additional SIP Message Information fields. Retrievable Call Information table (pg. 78): Revised datatype for H.323 Call ID and added info on SIP Call ID Examples of Retrieving Call-Related Information section (pg. 81): Added code examples for retrieving and pasrsing Call ID. Supported SIP Message Information Fields table (pg. 93): Added entries for Call ID, Diversion URI, Referred-by, and Replaces. Updated Contact URI entry to indicate setting is supported. Nonstandard Registration Message section (pg. 104): Corrected parameters and added code example gc_GetCallInfo( ) Variances for IP section (pg. 166): Added information on getting Call ID. Added SIP-specific address formats (To URI and From URI) gc_MakeCall( ) Variances for IP section (pg. 172): Added note about SIP timeout Configurable Call Parameters When Using H.323 table (pg. 173): Corrected value names for IPPARM_CONNECTIONMETHOD. Added entry for IPSET_CALLINFO/IPPARM_CALLID. Configurable Call Parameters When Using SIP table (pg. 176): Corrected value names for IPPARM_CONNECTIONMETHOD. Added entry for IPSET_CALLINFO/IPPARM_CALLID. gc_Start( ) Variances for IP section (pg. 201): Added information on default board instances and parameter values Summary of Parameter Sets and Parameter Usage table (pg. 209): Updated info for IPSET_CALLINFO/IPPARM_CALLID. Added entries for IPSET_SIP_MSGINFO/IPPARM_CALLID_HDR, IPSET_SIP_MSGNFO/IPPARM_DIVERSION_URI, IPSET_SIP_MSGINFO/IPPARM_REFERRED_BY, and IPSET_SIP_MSGINFO/IPPARM_REPLACES. Added set/send info for IPSET_SIP_MSGINFO/IPPARM_CONTACT_URI. IPSET_CALLINFO Parameter Set table (pg. 217): Updated type and description for IPPARM_CALLID. Corrected value names for IPPARM_CONNECTIONMETHOD. IPSET_SIP_MSGINFO Parameter Set table (pg. 228): Added entries for IPPARM_CALLID_HDR, IPPARM_DIVERSION_URI, IPPARM_REFERRED_BY, and IPPARM_REPLACES Updated IPPARM_CONTACT_URI to indicate that setting is supported. Added length defines for all parameters. IP_VIRTBOARD structure description (pg. 243): Added default values to field descriptions

Document No.	Publication Date	Description of Revisions
05-2039-002	April 2004	Summary of Call-Related Information that can be Set table: Added entries for Call ID, MediaWaitForConnect, and PresentationIndicator. Coders Supported for Host Media Processing (HMP) table: Corrected G.711 entries to indicate VAD must be disabled (PTR 32576). Added row for G.729a. Corrected frame size for G.729a+b. Added row for T.38. (PTR 32623) Setting Busy Reason Codes: New section. Example of Retrieving Call-Related Information section: Corrected both example programs Getting Notification of DTMF Detection section: Removed description of unsupported IPPARM_DTMF_RFC_2833 parameter Generating DTMF section: Removed description of IPPARM_DTMF_RFC_2833 parameter Generating or Detecting DTMF Tones Using a Voice Resource: New section Enabling and Disabling Unsolicited Notification Events section: Removed description of unsupported EXTENSIONEVT_DTMF_RFC2833 parameter Setting QoS Threshold Values and Retrieving QoS Threshold Values: Corrected ParmSetID name in both code examples (PTR 32690) Registration section: Removed incorrect reference to LRQ/LCF/LRJ RAS messages; corrected code example for SIP registration; added table to map abstract registrar registration concepts to SIP REGISTER elements Gatekeeper Registration Failure: New section. Global Call Functions Supported by IP section: Added bullet to indicate support for gc_GetCTInfo( ) gc_GetCTInfo( ) Variances for IP section: New section gc_MakeCall( ) Variances for IP section: Clarified procedure for setting protocol to use on multi-protocol devices. Added information to Forming a Destination Address String section about specifying port address in TCP/IP destination addresses. gc_ReqService( ) Variances for IP section: Added SIP support for alias gc_SetUserInfo( ) Variances for IP section: Added note about not using this function to set protocol to use on multi-protocol devices. gc_Start( ) Variances for IP sectio: Added note regarding network adaptor enabling/disabling. Added information about initialization functions and overriding defaults when appropriate. Initialization Functions: New section Summary of Parameter Sets and Parameter Usage table: Updated IPSET_CALLINFO/IPPARM_CALLID entry to include setting and sending Call ID Added IPSET_CALLINFO/IPPARM_MEDIAWAITFORCONNECT parameter Added IPSET_CALLINFO/IPPARM_PRESENTATION_IND parameter Added IPSET_CALLINFO/IPPARM_PROGRESS_IND parameter Added IPSET_H323_RESPONSE_CODE/IPPARM_BUSY_CAUSE parameter Updated IPSET_LOCAL_ALIAS set entries to add SIP support Added IPSET_SIP_RESPONSE_CODE/IPPARM_BUSY_REASON parameter Removed gc_SetConfigData( ) from list of functions that can be used to set TOS Removed description of unsupported IPPARM_DTMF_RFC_2833 parameter Parameter Set Reference section: Added and updated data type and size information for all parameter sets in section

Document No.	Publication Date	Description of Revisions
05-2039-002 (continued)		IPSET_CALLINFO section: Added entries for new parameters IPPARM_MEDIAWAITFORCONNECT, IPPARM_PRESENTATION_IND, and IPPARM_PROGRESS_IND IPSET_DTMF section: Removed description of unsupported IPPARM_DTMF_RFC_2833 parameter IPSET_EXTENSIONEVT_MSK section: Removed description of unsupported EXTENSIONEVT_DTMF_RFC2833 parameter IPSET_H323_RESPONSE_CODE: New section IPSET_REG_INFO section: Added row for IPPARM_REG_TYPE IPSET_SIP_MSGINFO section: Added section for parameters used when setting and retrieving SIP Message Information fields IPSET_SIP_RESPONSE_CODE: New section IP_REGISTER_ADDRESS structure description: corrected description of time_to_live field IP_RFC2833_EVENT structure description: Removed as unsupported IP_VIRTBOARD structure description: Updated to refer to INIT_IP_VIRTBOARD() initialization function. Added sup_serv_mask, h323_msginfo_mask, and terminal_type fields (PTR 30491) IPADDR structure description: Added note that only supported ipv4 field value is IP_CFG_DEFAULT. Added info about byte order for IPv4 addresses. IPCCLIB_START_DATA structure description: Updated to refer to INIT_ IPCCLIB_START_DATA() initialization function. IP-Specific Event Cause Codes chapter: Updated descriptions of the possible event causes (PTR 31213)
05-2039-001	September 2003	Initial production version of document. Much of the information contained in this document was previously published in the *Global Call IP over Host-based Stack Technology User's Guide*, document number 05-1512-005. Major changes as compared to 05-1512-005 include: Added T.38 Fax Server support Added Call Transfer support when using H.450.2 Added information about accessing SIP message information fields Specific changes include: Call Transfer Glare Condition: Added section and scenario diagram Specifying DTMF Support: Changed description of how to use GC_PARM_BLK to discover which DTMF modes are supported. Added information to IP-Specific Function Information for call transfer functions: gc_AcceptInitXfer( ) gc_AcceptXfer( ) gc_InitXfer( ) gc_InvokeXfer( ) gc_RejectInitXfer( ) gc_RejectXfer( ) gc_Start( ) Variances for IP: Added note regarding use of IP_CFG_MAX_AVAILABLE_CALLS

About This Publication

The following topics provide information about this publication.

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

Purpose

This guide is for users of the Global Call API writing applications that use host-based IP H.323 or SIP 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 API Programming Guide* and the *Global Call API Library Reference*, which describe the generic behavior of the Global Call API.

This publication specifically documents the Global Call API as it is implemented in the Intel® NetStructure™ Host Media Processing Software 1.2 for Linux* release. The Global Call API implementation in Intel® Dialogic® System Release software is documented in a separate set of documents.

Intended Audience

This guide is intended for:

- System IntegratorsIndependent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

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

How to Use This Publication

This guide is divided into the following chapters:

- Chapter 1, "IP Overview" gives a overview of VoIP technology and brief introductions to the H.323 and SIP standards for novice users.
- Chapter 2, "Global Call Architecture for IP" describes how Global Call can be used with IP technology and provides an overview of the architecture.

- Chapter 3, "IP Call Scenarios" provides some call scenarios that are specific to IP technology.
- Chapter 4, "IP-Specific Operations" describes how to use Global Call to perform IP-specific operations, such as setting call related information, registering with a registration server, etc.
- Chapter 5, "Building Global Call IP Applications" provides guidelines for building Global Call applications that use IP technology.
- Chapter 6, "Debugging Global Call IP Applications" provides information for debugging Global Call IP applications.
- Chapter 7, "IP-Specific Function Information" describes the additional functionality of specific Global Call functions used with IP technology.
- Chapter 8, "IP-Specific Parameters" provides a reference for IP-specific parameter set IDs and their associated parameter IDs.
- Chapter 9, "IP-Specific Data Structures" provides a data structure reference for Global Call IP-specific data structures.
- Chapter 10, "IP-Specific Event Cause Codes" describes IP-specific event cause codes.
- Chapter 11, "Supplementary Reference Information" provides supplementary information including technology references and formats for called and calling party addresses for H.323.
- A Glossary and an Index can be found at the end of the document.

Related Information

Refer to the following documents and web sites for more information about developing IP telephony applications that use the Global Call API:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *IP Media Library API Programming Guide*
- *IP Media Library API Library Reference*
- ITU-T Recommendation H.323, Packet-based multimedia communications systems, *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.323*
- ITU-T Recommendation H.225.0, Call signalling protocols and media stream packetization for packet-based multimedia communication systems, *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.225.0*
- ITU-T Recommendation H.245, Control protocol for multimedia communication, *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.245*
- ITU-T Recommendation H.450.2, Call transfer supplementary service for H.323, *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.450.2*
- Internet Engineering Task Force (IETF) Request for Comments RFC3261, *SIP: Session Initiation Protocol*, *ftp://ftp.rfc-editor.org/in-notes/rfc3261.txt*
- Internet Engineering Task Force (IETF) Request for Comments RFC1889, *RTP: A Transport Protocol for Real-Time Applications*, *ftp://ftp.rfc-editor.org/in-notes/rfc3261.txt*
- *http://developer.intel.com/design/telecom/support* (for technical support)
- *http://www.intel.com/design/network/products/telecom* (for product information)

IP Overview 1

This chapter provides overview information about the following topics:

1.1 Introduction to VoIP

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, Telecommunications Standardization Sector (ITU-T), which has defined the following:
  - Recommendation H.323, covering Packet-based Multimedia Communications Systems (including VoIP)
- Internet Engineering Task Force (IETF), which has defined drafts of the several RFC (Request for Comment) documents, including the following:
  - RFC 3261, the Session Initiation Protocol (SIP)

The H.323 recommendation was developed in the mid 1990s and is a mature protocol.

SIP (Session Initiation Protocol) is an emerging protocol for setting up telephony, conferencing, multimedia, and other types of communication sessions on the Internet.

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). This section discusses the following topics about H.323:

- H.323 Entities
- H.323 Protocol Stack
- Codecs
- Basic H.323 Call Scenario

- Registration with a Gatekeeper
- H.323 Call Scenario via a Gateway

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 multimedia coders/decoders (codecs) and setup and control signaling.

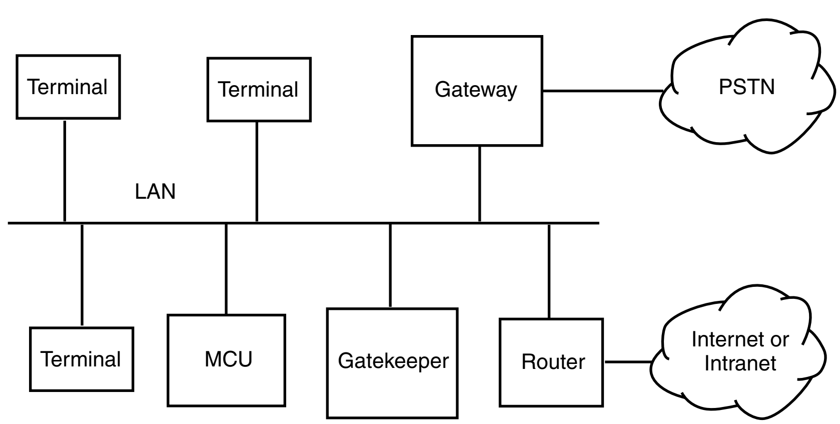
Gateway
: Provides the interface between a packet-based network (for example, an IP network) and a circuit-switched network (for example, the PSTN). A gateway translates communication procedures and formats between networks. It handles call setup and teardown and the compression and packetization of 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 supports 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.

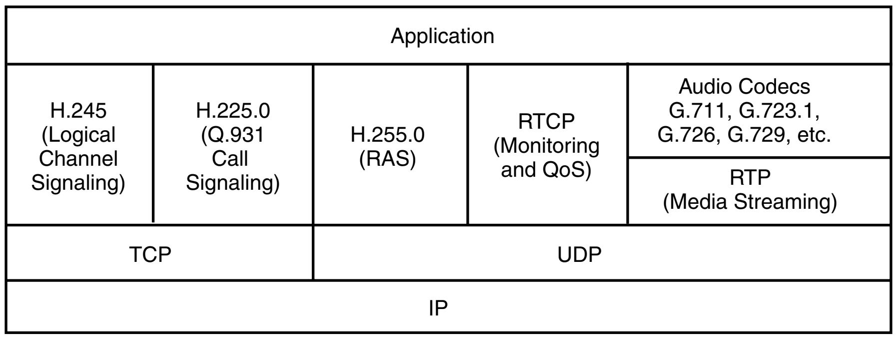
Figure 1. Typical H.323 Network



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, the packetization and synchronization of media streams, and Registration, Admission, and Status (RAS).

Real Time Protocol (RTP)
: The RTP specification is an IETF draft standard (RFC 1889) 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 IETF RTP specification (RFC 1889) 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 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 kbps to 64 kbps. 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 kbps.

G.723.1
: Provides bit rates of either 5.3 or 6.4 kbps. Voice communication using this codec typically exhibits some form of degradation.

G.729
: Provides a bit rate of 8 kbps. This codec is very popular for voice over frame relay and for V.70 voice and data modems.

GSM
: Provides a bit rate of 13 kbps. This codec is based on a telephony standard defined by the European Telecommunications Standards Institute (ETSI). The 13 kbps bit rate is achieved with little degradation of voice-grade audio.

1.2.4 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 scenario described in this section assumes a slow start connection procedure. See Section 4.2, "Using Fast Start and Slow Start Setup", on page 66 for more information on the difference between the slow start and fast start connection procedure.

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.

Call Setup

Establishing a call between two endpoints requires two TCP connections between the endpoints:

- One for the call setup (Q.931/H.225 messages)
- One for capability exchange and call control (H.245 messages)

Note: It is also possible to encapsulate H.245 media control messages within Q.931/H.225 signaling messages. The concept is known as *H.245 tunneling*. If tunneling is enabled, one less TCP port is required for incoming connections.

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 along with 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.

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.

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. The scenario described here assumes a slow start connection procedure. See Section 4.2, "Using Fast Start and Slow Start Setup", on page 66 for more information on the difference between the slow start and fast start connection procedure.

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 indicate which port endpoint A will receive RTP data and send RTCP reports to. Once these ports have been identified, the next phase of the call scenario, Data Exchange, can begin.

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.

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.

Either endpoint (but typically the endpoint that initiates the termination) then sends an H.225.0 ReleaseComplete message over the call signalling channel, which closes that channel and ends the call.

1.2.5 Registration with a Gatekeeper

In a H.323 network, a gatekeeper is an entity that can manage all endpoints that can send or receive calls. Each gatekeeper controls a specific zone and endpoints must register with the gatekeeper to become part of the gatekeeper's zone. The gatekeeper provides call control services to the endpoints in its zone. The primary functions of the gatekeeper are:

- address resolution by translating endpoint aliases to transport addresses
- admission control for authorizing network access
- bandwidth management
- network management (in routed mode)

Endpoints communicate with a gatekeeper using the Registration, Admission, and Status (RAS) protocol. A RAS channel is an unreliable channel that is used to carry RAS messages (as described in the H.255 standard). The RAS protocol covers the following:

- Gatekeeper Discovery
- Endpoint Registration
- Endpoint Deregistration
- Endpoint Location
- Admission, Bandwidth Change and Disengage

Note: The RAS protocol covers status request, resource availability, nonstandard registration messages, unknown message response and request in progress that are not described in any detail in this overview. See *ITU-T Recommendation H.225.0 (09/99)* for more information.

Gatekeeper Discovery

An endpoint uses a process called *gatekeeper discovery* to find a gatekeeper with which it can register. To start this process, the endpoint can multicast a GRQ (gatekeeper request) message to the well-known discovery multicast address for gatekeepers. One or more gatekeepers may respond with a GCF (gatekeeper confirm) message indicating that it can act as a gatekeeper for the endpoint. If a gatekeeper does not want to accept the endpoint, it returns GRJ (gatekeeper reject). If more than one gatekeeper responds with a GCF message, the endpoint can choose which gatekeeper it wants to register with. In order to provide redundancy, a gatekeeper may specify an alternate gatekeeper in the event of a failure in the primary gatekeeper. Provision for the alternate gatekeeper information is provided in the GCF and RCF messages.

Endpoint Registration

An endpoint uses a process called *registration* to join the zone associated with a gatekeeper. In the registration process, the endpoint informs the gatekeeper of its transport, alias addresses, and endpoint type. Endpoints register with the gatekeeper identified in the gatekeeper discovery process described above. Registration can occur before any calls are made or periodically as necessary. An endpoint sends an RRQ (registration request) message to perform registration and in return receives an RCF (registration confirmation) or RRJ (registration reject) message.

Endpoint Deregistration

An endpoint may send an URQ (unregister request) in order to cancel registration. This enables an endpoint to change the alias address associated with its transport address or vice versa. The gatekeeper responds with an UCF (unregister confirm) or URJ (unregister reject) message.

The gatekeeper may also cancel an endpoint's registration by sending a URQ (unregister request) to the endpoint. The endpoint should respond with an UCF (unregister confirm) message. The endpoint should then try to re-register with a gatekeeper, perhaps a new gatekeeper, prior to initiating any calls.

Endpoint Location

An endpoint that has an alias address for another endpoint and would like to determine its contact information may issue a LRQ (location request) message. The LRQ message may be sent to a specific gatekeeper or multicast to the well-known discovery multicast address for gatekeepers. The gatekeeper to which the endpoint to be located is registered will respond with an LCF (location confirm) message. A gatekeeper that is not familiar with the requested endpoint will respond with LRJ (location reject).

Admission, Bandwidth Change and Disengage

The endpoint and gatekeeper exchange messages to provide admission control and bandwidth management functions. The ARQ (admission request) message specifies the requested call bandwidth. The gatekeeper may reduce the requested call bandwidth in the ACF (admission confirm) message. The ARQ message is also used for billing purposes, for example, a gatekeeper may respond with an ACF message just in case the endpoint has an account so the call can be charged. An endpoint or the gatekeeper may attempt to modify the call bandwidth during a call using a BRQ (bandwidth change request) message. An endpoint will send a DRQ (disengage request) message to the gatekeeper at the end of a call.

1.2.6 H.323 Call Scenario via a Gateway

While the call scenario described in Section 1.2.4, "Basic H.323 Call Scenario", on page 20 is useful for explaining the fundamentals of an H.323 call, it is not a realistic call scenario. Most significantly, the IP addresses of both endpoints were defined to be known in the example, while 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

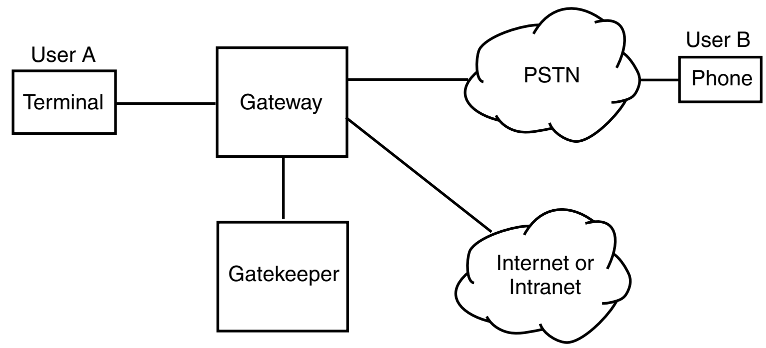


Figure 3 also shows a gatekeeper. The gatekeeper provides network services such as Registration, Admission, and Status (RAS) and address mapping. When a gatekeeper is present, all endpoints managed by the gatekeeper must register with the gatekeeper at startup. The gatekeeper tracks which endpoints are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

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.

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 case, 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 and Data Exchange, can begin.

Call Signaling and Data Exchange

Endpoint A can now send 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 gateway 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 the Capability Exchange, Call Initiation, and Data Exchange phases in Section 1.2.4, "Basic H.323 Call Scenario", on page 20 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.

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.

1.3 SIP Overview

Session Initiation Protocol (SIP) is an ASCII-based, peer-to-peer protocol designed to provide telephony services over the Internet. The SIP standard was developed by the Internet Engineering Task Force (IETF) and is one of the most commonly used protocols for VoIP implementations. This section discusses the following topics about SIP:

- Advantages of Using SIP
- SIP User Agents and Servers
- Basic SIP Operation
- Basic SIP Call Scenario
- SIP Messages

1.3.1 Advantages of Using SIP

Some of the advantages of using SIP include:

- The SIP protocol stack is smaller and simpler than other commonly used VoIP protocols, such as H.323.
- SIP-based systems are more easily scalable because of the peer-to-peer architecture used. The hardware and software requirements for adding new users to SIP-based systems are greatly reduced.
- Functionality is distributed over different components. Control is decentralized. Changes made to a component have less of an impact on the rest of the system.
- SIP is Internet-enabled.

1.3.2 SIP User Agents and Servers

User agents (UAs) are appliances or applications, such as SIP phones, residential gateways and software that initiate and receive calls over a SIP network.

Servers are application programs that accept requests, service requests and return responses to those requests. Examples of the different types of servers are:

Location Server
: Used by a SIP redirect or proxy server to obtain information about the location of the called party.

Proxy Server
: An intermediate program that operates as a server and a client and which makes requests on behalf of the client. A proxy server does not initiate new requests, it interprets and possibly modifies a request message before forwarding it to the destination.

Redirect Server
: Accepts a request from a client and maps the address to zero or more new addresses and returns the new addresses to the client. The server does not accept calls or generate SIP requests on behalf of clients.

Registrar Server

Accepts REGISTER requests from clients. Often, the registrar server is located on the same physical server as the proxy server or redirect server.

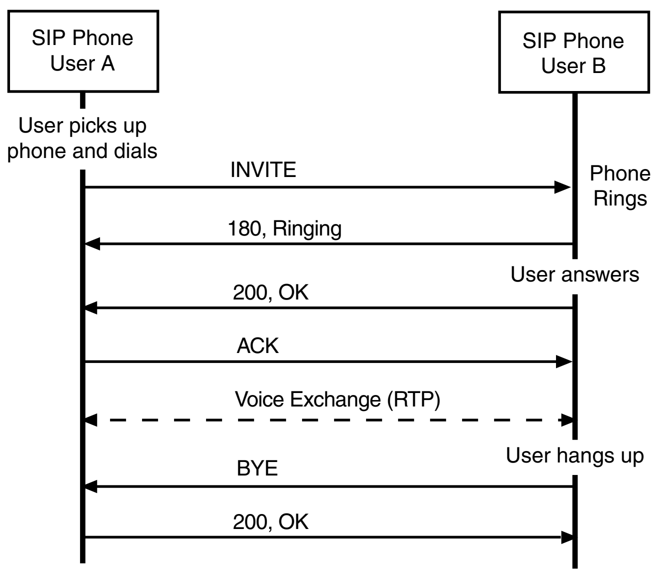
1.3.3 Basic SIP Operation

Callers and callees are identified by SIP addresses. When making a SIP call, a caller first locates the appropriate server and then sends a SIP request. The most common SIP operation is the invitation request. Instead of directly reaching the intended callee, a SIP request may be redirected or may trigger a chain of new SIP requests by proxies. Users can register their location(s) with SIP servers.

1.3.4 Basic SIP Call Scenario

Figure 4 shows the basic SIP call establishment and teardown scenario.

Figure 4. Basic SIP Call Scenario



1.3.5 SIP Messages

In SIP, there are two types of messages:

- SIP Request Messages
- SIP Response Messages

SIP Request Messages

The most commonly used SIP request messages are:

- INVITE
- ACK
- BYE
- REGISTER
- CANCEL
- OPTIONS

For more information, see RFC 3261 at *http://www.ietf.org/rfc/rfc3261.txt?number=3261*.

SIP Response Messages

SIP response messages are numbered. The first digit in each response number indicates the type of response. The response types are as follows:

1xx
: Information responses; for example, 180 Ringing

2xx
: Successful responses; for example, 200 OK

3xx
: Redirection responses; for example, 302 Moved Temporarily

4xx
: Request failure responses; for example, 402, Forbidden

5xx
: Server failure responses; for example, 504, Gateway Timeout

6xx
: Global failure responses; for example, 600, Busy Everywhere

For more information, see RFC 3261 at the URL given above.

Global Call Architecture for IP 2

This chapter discusses the following topics:

2.1 Global Call over IP Architecture with a Host-Based Stack

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:

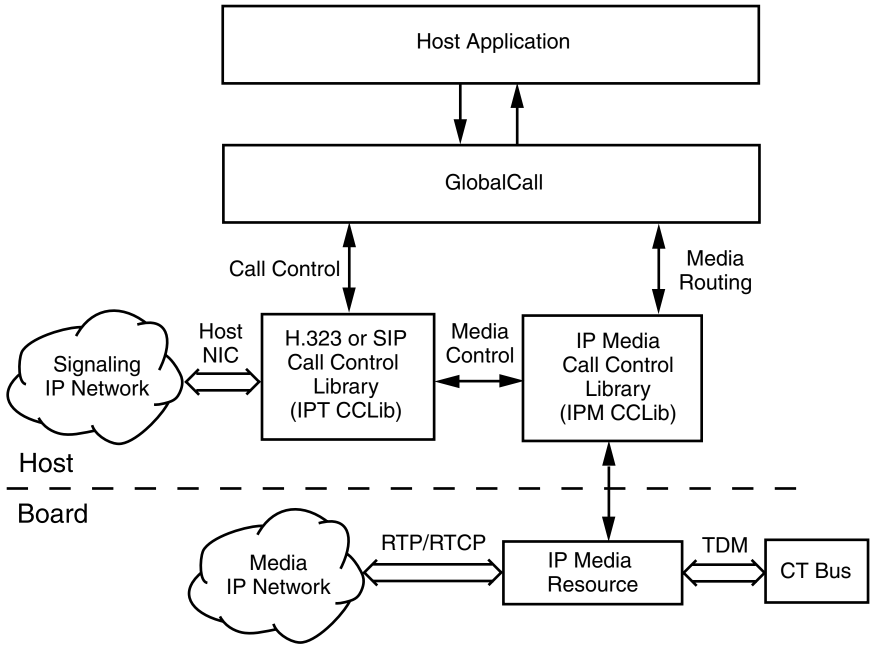
- call control capabilities for establishing calls over an IP network
- support for IP Media control by providing the ability to open and close IP Media channels for streaming

Global Call supports a system configuration where the IP signaling stack runs on the host and a Host Media Processing virtual board provides the IP resources for media processing.

Note: Global Call supports the RADVISION* H.323 and SIP stacks. If other third-party call control stacks are used, Global Call cannot be used for IP call control, but the IP Media Library can be used for media resource management. See the *IP Media Library API Programming Guide* and *IP Media Library API Library Reference* for more information.

Figure 5 shows the Global Call over IP architecture when using a DM3 board or an Intel NetStructure IPT board and a host-based stack provided with the system software

Figure 5. Global Call over IP Architecture Using a Host-Based Stack



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.

2.2 Architecture Components

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

- Host Application
- Global Call
- IP Signaling Call Control Library (IPT CCLib)
- IP Media Call Control Library (IPM CCLib)
- IP Media Resource

2.2.1 Host Application

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

- initializes Global Call
- opens and closes IP line devices
- opens and closes IP Media devices
- opens and closes PSTN 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 PSTN alarms and errors

2.2.2 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

2.2.3 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 and/or SIP 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

2.2.4 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

2.2.5 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 IP network
- decodes IP packets received from the IP network into PCM data transmitted to the TDM bus
- configures and reports QoS information to the IP Media stream

2.3 Device Types and Usage

This section includes information about device types and usage:

- Device Types Used with IP
- IPT Board Devices
- IPT Network Devices
- IPT Start Parameters

2.3.1 Device Types Used with IP

When using Global Call with IP technology, a number of different device types are used:

IPT Board Device
: A virtual entity that represents a NIC or NIC address (if one NIC supports more than one IP address). The format of the device name is **iptBx**, where **x** is the logical board number that corresponds to the NIC or NIC address. See Section 2.3.2, "IPT Board Devices", on page 35 for more information.

IPT Network Device
: Represents a logical channel over which calls can be made. This device is used for call control (call setup and tear down). The format of the device name is **iptBxTy**, where **x** is the logical board number and **y** is the logical channel number. See Section 2.3.3, "IPT Network Devices", on page 36 for more information.

IP Media Device
: Represents a media resource that is used to control RTP streaming, monitoring Quality of Service (QoS) and the sending and receiving of DTMF digits. The format of the device name is **ipmBxCy**, where **x** is the logical board number and **y** is the logical channel number.

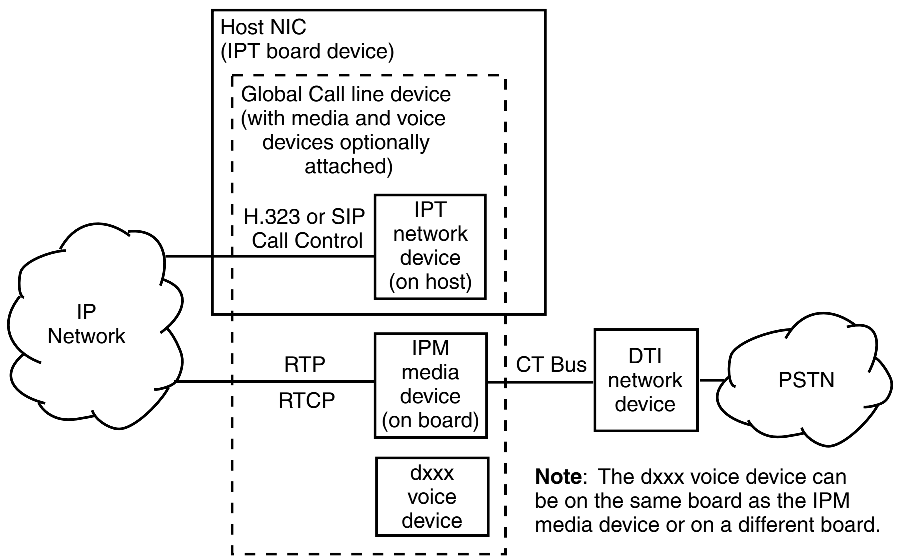
The IPT network device (iptBxTy) and the IP Media device (ipmBxCy) can be opened simultaneously in the same **gc_OpenEx( )** command. If a voice resource is available in the system, for example an IP board that provides voice resources or any other type of board that provides voice resources, a voice device can also be included in the same **gc_OpenEx( )** call to provide voice capabilities on the logical channel. See Section 7.2.17, "gc_OpenEx( ) Variances for IP", on page 187 for more information.

Alternatively, the IPT network device (iptBxTy) and the IP Media device (ipmBxCy) can be opened in separate **gc_OpenEx( )** calls and subsequently attached using the **gc_AttachResource( )** function.

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 4.15, "Quality of Service Alarm Management", on page 108 for more information.

Figure 6 shows the relationship between the various types of Global Call devices when a single Host NIC is used.

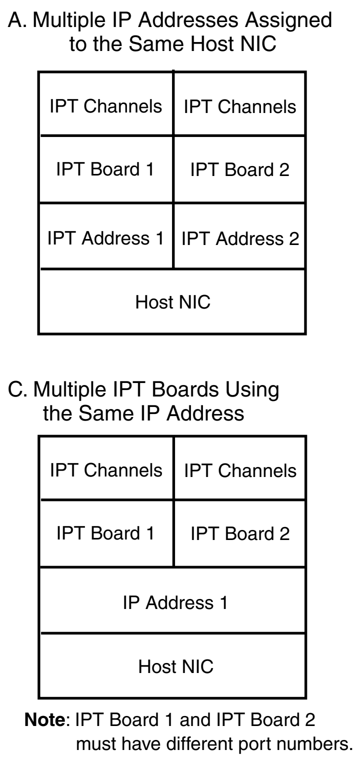
Figure 6. Global Call Devices



2.3.2 IPT Board Devices

An IPT board device is a virtual entity that corresponds to an IP address and is capable of handling both H.323 and SIP protocols. The application uses the **gc_Start( )** function to bind IP addresses to IPT virtual board devices. Possible configurations are shown in Figure 7. The operating system must support the IP address and underlying layers before the Global Call application can take advantage of the configurations shown in Figure 7. Up to eight virtual IPT boards can be configured in one system. For each virtual IPT board, it is possible to configure the local address and signaling port (H.323 and SIP), the number of IPT network devices that can be opened simultaneously, etc. See Section 7.2.26, "gc_Start( ) Variances for IP", on page 201 for more information on how to configure IPT board devices.

Figure 7. Configurations for Binding IPT Boards to NIC IP Addresses



Once the IPT board devices are configured, the application can open line devices with the appropriate IPT network device (IPT channel) and optionally IPT Media device (IPM channel).

The **gc_SetConfigData( )** function can be used on an IPT board device to apply parameters to all IPT channels associated with the IPT board device. The application can use the **gc_AttachResource( )** and **gc_Detach( )** functions to load balance which host NIC makes a call for a particular IPT media device (IPM channel). It is also possible that the operating system can perform load balancing using the appropriate NIC for call control as shown in Figure 7, configuration D.

The **gc_ReqService( )** function is used on an IPT board device for registration with an H.323 gatekeeper or SIP registrar. See Section 7.2.21, "gc_ReqService( ) Variances for IP", on page 190 for more information.

2.3.3 IPT Network Devices

Global Call supports three types of IPT network devices:

- H.323 only (P_H323 in the **devicename** string when opening the device)
- SIP only (P_SIP in the **devicename** string when opening the device)
- Dual protocol, H.323 and SIP (P_IP in the **devicename** string when opening the device)

The device type is determined when using the **gc_OpenEx( )** function to open the device. H.323 and SIP only devices are capable of initiating and receiving calls of the selected protocol type only.

Dual protocol devices are capable of initiating and receiving calls using either the H.323 or SIP protocol. The protocol used by a call on a dual protocol device is determined during call setup as follows:

- for outbound calls, by a parameter to the **gc_MakeCall( )** function
- for inbound calls, by calling **gc_GetCallInfo( )** to retrieve the protocol type used. In this case, the application can query the protocol type of the current call after the call is established, that is, as soon as either GCEV_DETECTED (if enabled) or GCEV_OFFERED is received.

2.3.4 IPT Start Parameters

The application determines the number of boards that will be created by the IPT call control library (up to the number of available IP addresses). For each board, the host application will provide the following information:

- number of line devices on the board
- maximum number of IPT devices to be used for H.323 calls (used for H.323 stack allocation)
- maximum number of IPT devices to be used for SIP calls (used for SIP stack allocation)
- board IP address
- listen port for H.323
- listen port for SIP
- enable/disable access to SIP Message Information fields
- enable/disable call transfer when using the H.450.2 protocol
- enable/disable access to H.323 Message Information fields
- terminal type

IP Call Scenarios 3

This chapter provides common call control scenarios when using Global Call with IP technology. Topics include:

3.1 Basic Call Control Scenarios When Using IP Technology

This section provides details of the basic call control scenarios when using IP technology. The scenarios include:

- Basic Call Setup When Using H.323 or SIP
- Basic Call Teardown When Using H.323 or SIP

3.1.1 Basic Call Setup When Using H.323 or SIP

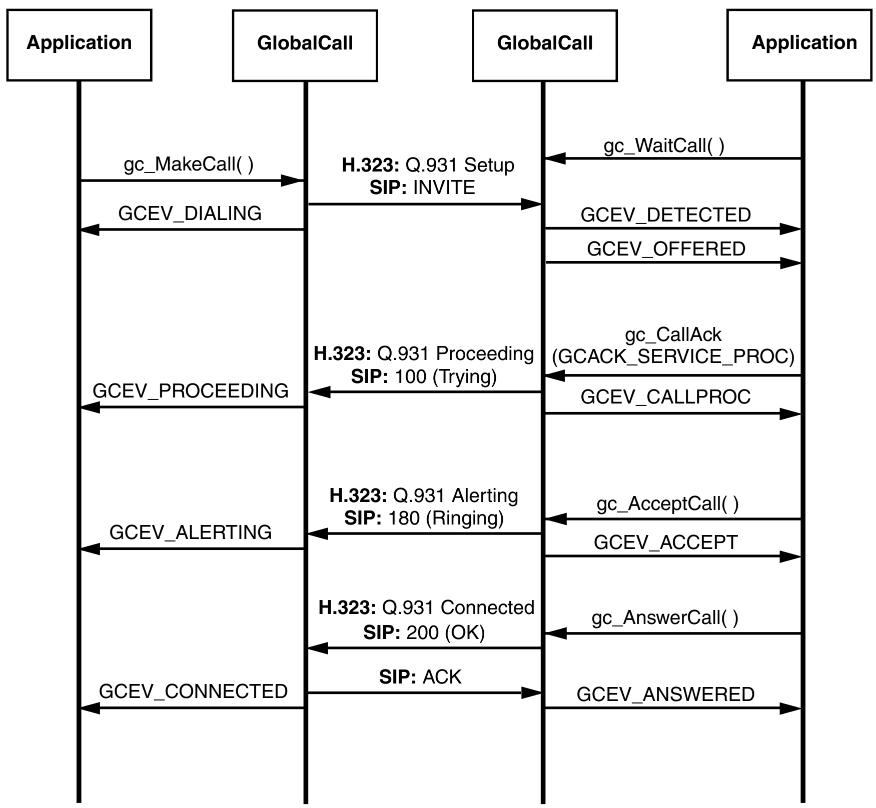
Figure 8 shows the basic call setup sequence when using H.323 or SIP.

Notes: *1.* This figure assumes that the network and media channels are already open and a media channel with the appropriate media capabilities is attached to the network channel. See Section 7.2.17, "gc_OpenEx( ) Variances for IP", on page 187 for information on opening and attaching network and media devices and Section 7.2.16, "gc_MakeCall( ) Variances for IP", on page 172 for detailed information on the specification of the destination address etc.

2. Only H.225.0 (Q.931) messages are shown in the sequence below. H.245 messages were omitted in the interest of simplification.

3. The destination address must be a valid address that can be translated by the remote node.

Figure 8. Basic Call Setup When Using H.323 or SIP

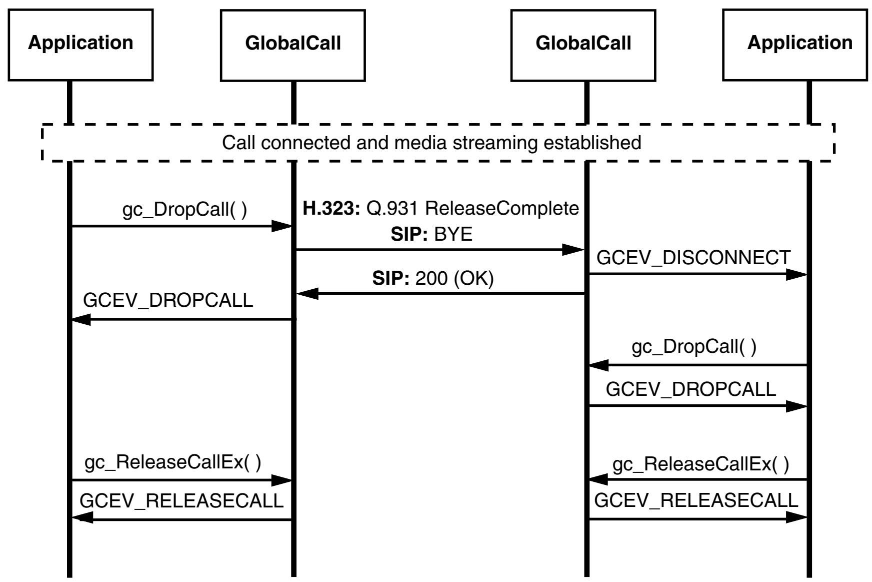


3.1.2 Basic Call Teardown When Using H.323 or SIP

Figure 9 shows the basic call teardown scenario when using Global Call with H.323 or SIP.

Note: Only H.225.0 (Q.931) messages are shown in the sequence below. H.245 messages were omitted in the interest of simplification.

Figure 9. Basic Call Teardown When Using H.323 or SIP



3.2 Call Transfer Scenarios When Using H.323

Global Call supports call transfer as described in the *Global Call API Programming Guide*. For technology-specific information, see Section 4.17, "Call Transfer When Using H.323", on page 122. The following scenarios demonstrate the call transfer capabilities provided when using H.323:

- General Conditions for Call Transfers
- Successful Blind Transfer Scenario
- Unsuccessful Blind Transfer Scenarios
- Successful Supervised Call Transfer Scenario
- Unsuccessful Supervised Transfer Scenarios

3.2.1 General Conditions for Call Transfers

When performing a call transfer operation, all involved call handles must be on the same stack instance. This imposes the following application restrictions for call transfer operations:

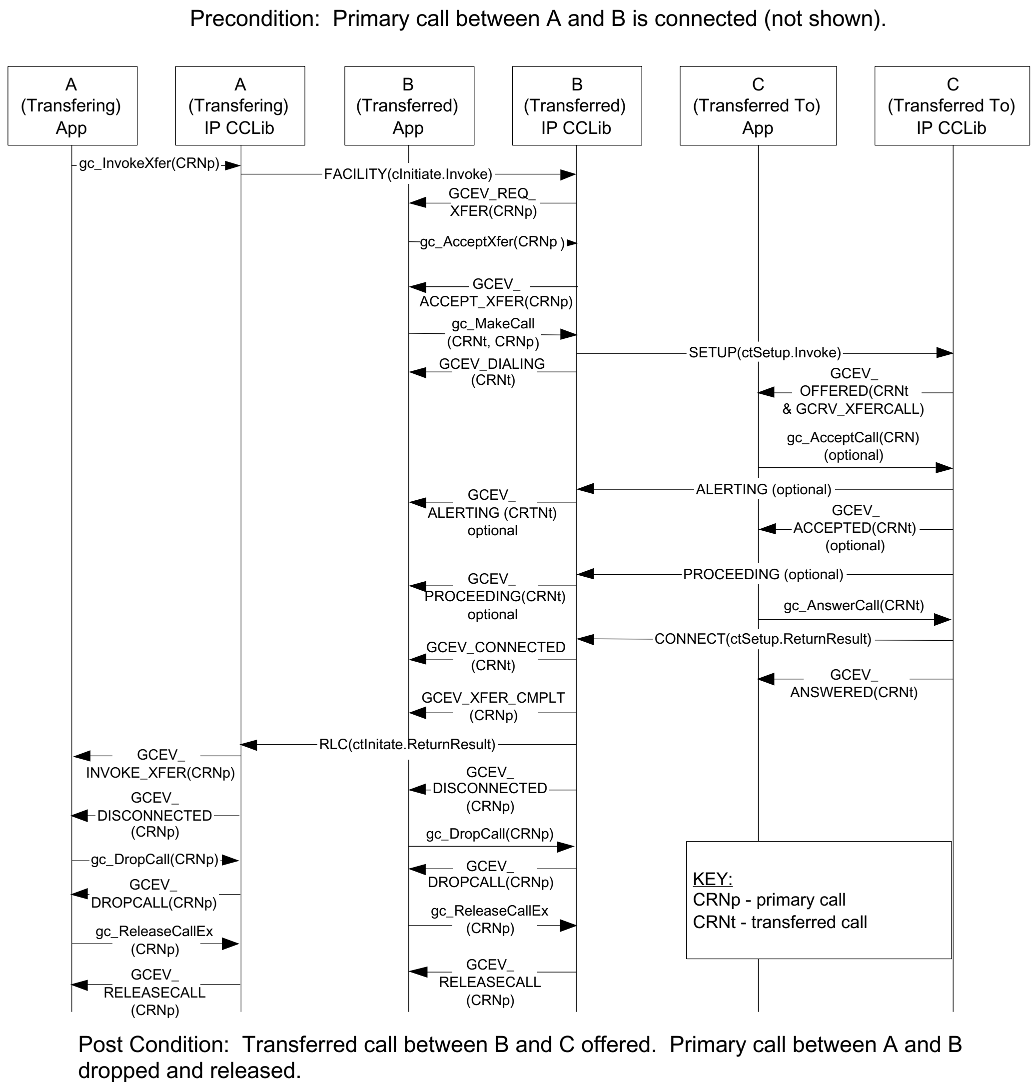
- When performing a supervised call transfer at party A, both the consultation line device and the transferring line device must be on the same virtual board.
- When performing a call transfer (either supervised or blind) at party B, both the transferring line device and the transferred line device must be on the same virtual board.
- When performing a supervised call transfer at party C, both the consultation line device and the transferred-to line device must be on the same virtual board.

3.2.2 Successful Blind Transfer Scenario

As indicated in Figure 10, the precondition for blind transfer is that the transferring endpoint (party A) and the transferred endpoint (party B) are participating in an active (primary) call and are in GCST_CONNNECTED from the perspective of the Global Call API. Completion of a successful blind transfer results in the eventual termination of the primary call, and the creation of the transferred call. Note that the connection of the transferred call is not a mandate for the completion of a blind transfer. It is always possible that the transferred call itself may possibly be left unanswered after ringing (ALERTING indication) and eventually abandoned and still be considered a *successful* blind transfer from the perspective of the transferring endpoint (party A). Successful blind transfer, in this regard requires only that some response notification was received, that is, either ALERTING or CONNECT, from the transferred-to endpoint.

For simplification purposes, Figure 10 does not indicate the opening and closing of logical channels (and the associated media sessions) as the control procedures are consistent with typical non-transfer related H.323 calls.

Figure 10. Successful Blind Transfer



Transferring Endpoint (Party A)

The transferring endpoint (party A) initiates the blind transfer via calling the **gc_InvokeTransfer( )** function, which results in the sending an ctInitiate.Invoke APDU in a FACILITY message. From this point forward, this endpoint is only subsequently notified as to the creation of the transferred call attempt. Note however, that it is not notified as to the end result of the transfer, specifically whether the transfer results in a connection or a no-answer. Instead, the transferring endpoint is only guaranteed notification that the transferred-to endpoint has been alerted to the incoming transferred call offering (that is, ringback). As specified in H.450.2, the ctInitiate.ReturnResult APDU may be returned in either ALERTING or CONNECT. The primary call will also be disconnected remotely via the transferred endpoint (party B) as part of a successful status notification from this endpoint. Both the forward and reverse logical channels will be closed along with their associated audio or data streams. From the Global Call API perspective, the primary call is terminated at the transferring endpoint, as indicated by the GCEV_DISCONNECTED event, implying the endpoint is then responsible for the drop and release of the primary call.

Transferred Endpoint (Party B)

The endpoint to be transferred (party B) is notified of the request to transfer from the initiating endpoint via the GCEV_REQ_XFER event. Assuming the party to be transferred accepts the transfer request via the **gc_AcceptXfer( )** function, it retrieves the destination address information from the unsolicited transfer request via the GC_REROUTING_INFO structure passed within the GCEV_REQ_XFER event. The endpoint to be transferred then uses the rerouting address information to initiate a call to the new destination party via **gc_MakeCall( )**. From the perspective of the application, this transferred call is treated in the same manner as a normal singular call and the party receives intermediate call state events as to the progress of the call (that is, GCEV_DIALING, GCEV_ALERTING, GCEV_PROCEEDING, and GCEV_CONNECTED). When the transferred endpoint receives its first indication from the transferred-to endpoint (party C) that the call transfer was successful (CTSetup.ReturnResult APDU), the transferred endpoint is notified of the transfer success and implicitly, without user or application initiation, disconnects the primary call with the transferring endpoint.

Assuming the transferred call is answered, the transferred endpoint is then involved in active media streaming with the transferred-to endpoint. Note that the notification of transfer success via the GCEV_XFERRED_CMPLT event may also arrive with any call progress events, that is, GCEV_ALERTING, GCEV_PROCEEDING, or GCEV_CONNECTED. Although the primary call to the transferring endpoint (party A) is implicitly dropped, the call itself must still be explicitly dropped via **gc_DropCall( )** to resynchronize the local state machine and released via **gc_ReleaseCallEx( )**.

Transferred-To Endpoint (Party C)

For the most part, from the perspective of the transferred-to endpoint (party C), the transferred call is treated as a typical incoming call. The call is first notified to the application via GCEV_DETECTED or GCEV_OFFERED events at which point the GCRV_XFERCALL cause value provided in the event will alert the application that this call offering is the result of a transfer. At that point, the application may retrieve the typical calling party information about the call. The transferred-to party is then provided the same methods of action as a typical incoming call, namely alerting the transferred endpoint (party B) that the call is proceeding (typical for gateways), ringback notification that the local user is being alerted, or simply answering the call. The only behavior change from this endpoint over typical non-transferred calls, is whether to treat or display the calling party information any differently if it is the result of a transfer. Assuming the transferred call is eventually connected or timed out on no answer, the transferred-to party must eventually drop and release this call as the case for non-transferred call.

3.2.3 Unsuccessful Blind Transfer Scenarios

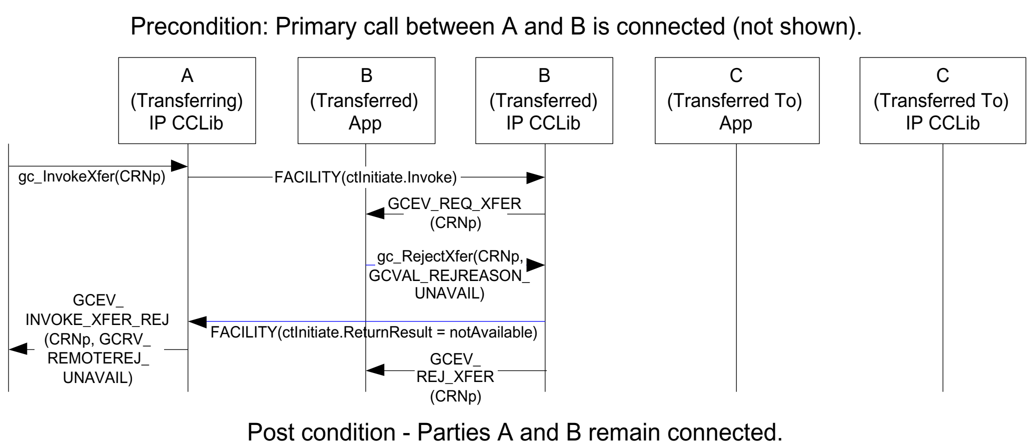
There are a several of scenarios where a blind call transfer may fail. The most common scenarios are described in the following topics:

- Party B Rejects Call Transfer
- No Response From Party B
- No Response From Party C
- Party B Clears Primary Call Before Transfer is Completed
- Party C is Busy When Transfer Attempted

3.2.3.1 Party B Rejects Call Transfer

As indicated in Figure 11, the application at the transferred endpoint (party B) may call the **gc_RejectXfer( )** function to signal via the ctInitiate.ReturnResult APDU that it cannot participate in a transfer. As a result, the GCEV_INVOKE_XFER_REJ termination event is received at transferring endpoint (party A) and the original primary call is left connected and in the GCST_CONNECTED state from the perspective of both A and B.

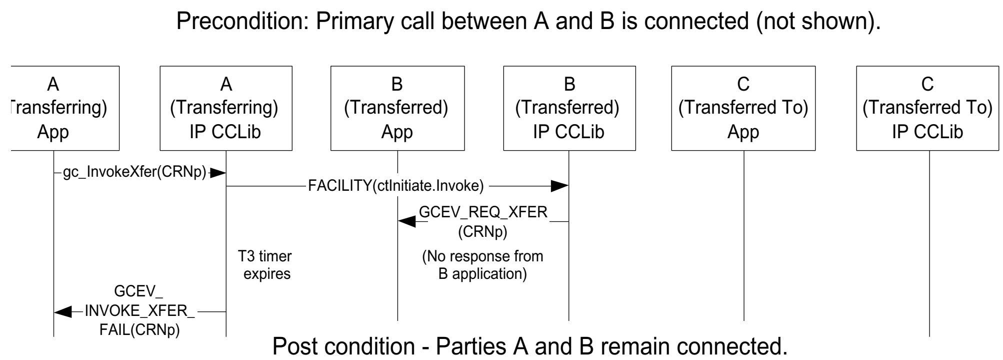
Figure 11. Blind Call Transfer Failure - Party B Rejects Call Transfer



3.2.3.2 No Response From Party B

As indicated in Figure 12, the transferred endpoint (party B) may not respond to the ctInitiate.ReturnResult APDU which would cause the T3 timer configured as 20 seconds at the transferring endpoint (party A) to expire. As a result, the GCEV_INVOKE_XFER_FAIL termination event would be received at transferring endpoint (party A) and the original primary call is left connected and in the GCST_CONNECTED state from the perspective of both A and B.

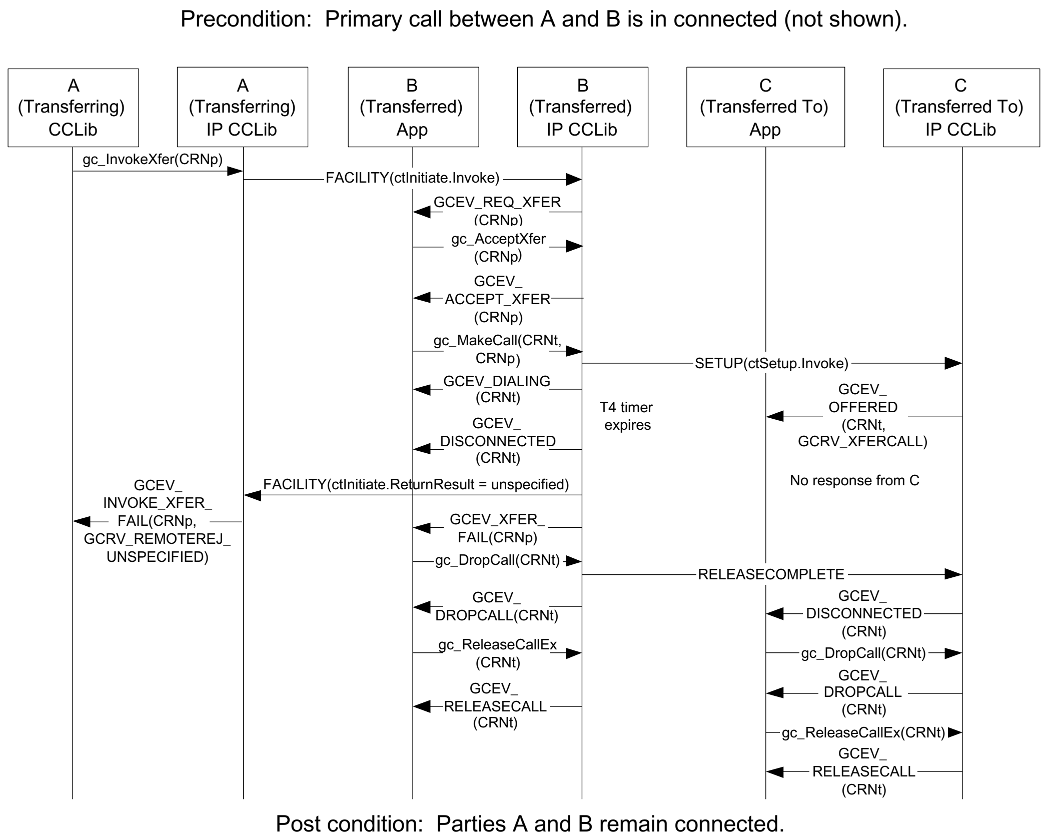
Figure 12. Blind Call Transfer Failure - No Response from Party B



3.2.3.3 No Response From Party C

As indicated in Figure 13, the transferred-to endpoint (party C) may not respond to the incoming call which would cause the T4 timer configured as 20 seconds at the transferred endpoint (party B) to expire. As a result, the transferred endpoint (party B) receives the GCEV_DISCONNECT event for the transferred call timeout and after sending a ctInitiate.ReturnResult = Unspecified APDU receives the GCEV_XFER_FAIL event on the primary call. Upon receiving the ctInitiate.ReturnResult = Unspecified APDU, the transferring endpoint (party A) is notified by the GCEV_INVOKE_XFER_FAIL termination event and the original primary call is left connected and in the GCST_CONNECTED state from the perspective of both A and B.

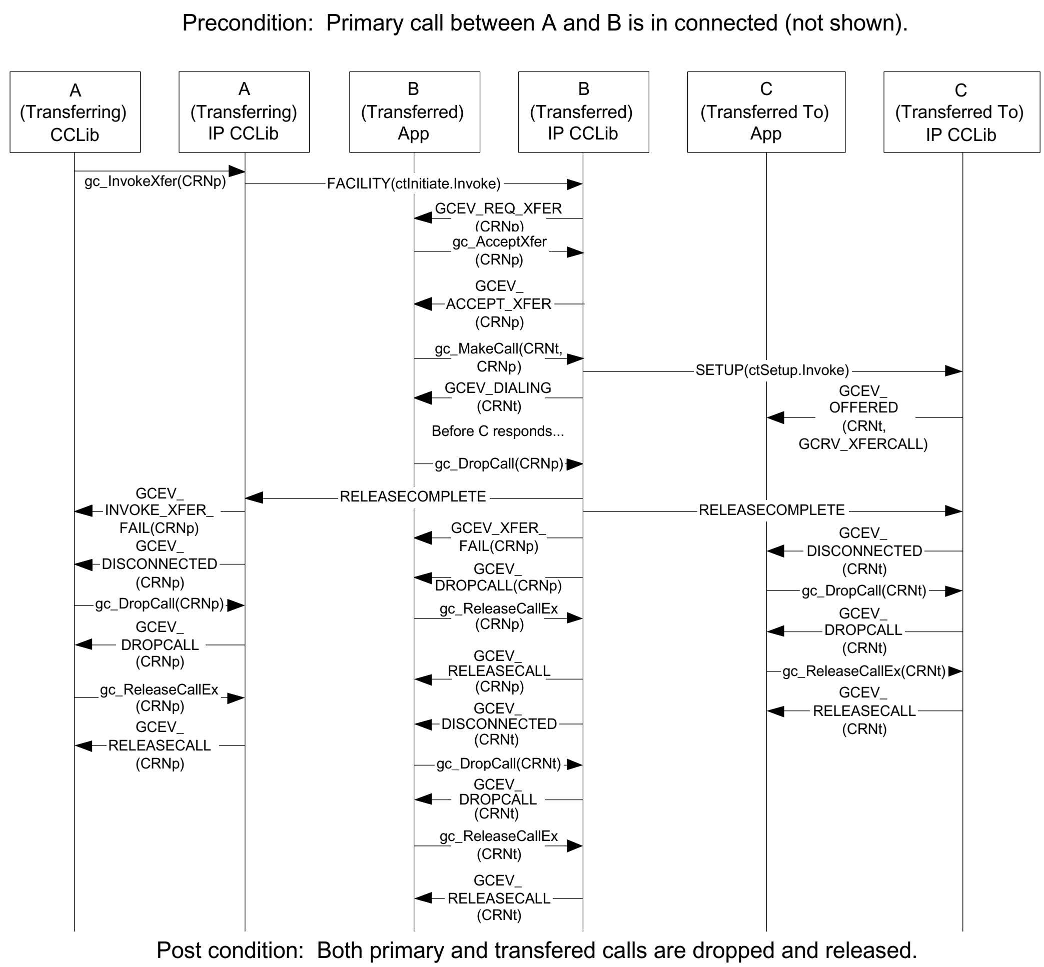
Figure 13. Blind Call Transfer Failure - No Response from Party C



3.2.3.4 Party B Clears Primary Call Before Transfer is Completed

The primary call may be cleared at any time while a blind transfer is in progress. As indicated in Figure 14, the transferred endpoint (party B) may clear the primary call while awaiting acknowledgement from the transferred-to endpoint (party C). As a result, the GCEV_INVOKE_XFER_FAIL termination event is received at transferring endpoint (party A) followed by a GCEV_DISCONNECT. Similarly, the GCEV_XFER_FAIL termination event is received at transferred endpoint (party B) followed by a GCEV_DROPCALL. At this point party A must drop and release the call while party B must release the call. The transferred call will also be abandoned implicitly per H.450.2 once the primary call is abandoned. The transferred-to endpoint will receive the GCEV_DISCONNECTED event at which point it must explicitly drop and release the abandoned transferred call attempt. Note that if instead party A initiated the clearing of the primary call while blind transfer is in progress, the only major difference with the corollary is that party B and not A would react to the primary disconnect (in the same manner as before) and explicitly drop the primary call; otherwise, the behavior is identical.

Figure 14. Blind Call Transfer Failure - Party B Clears Primary Call Before Transfer is Completed

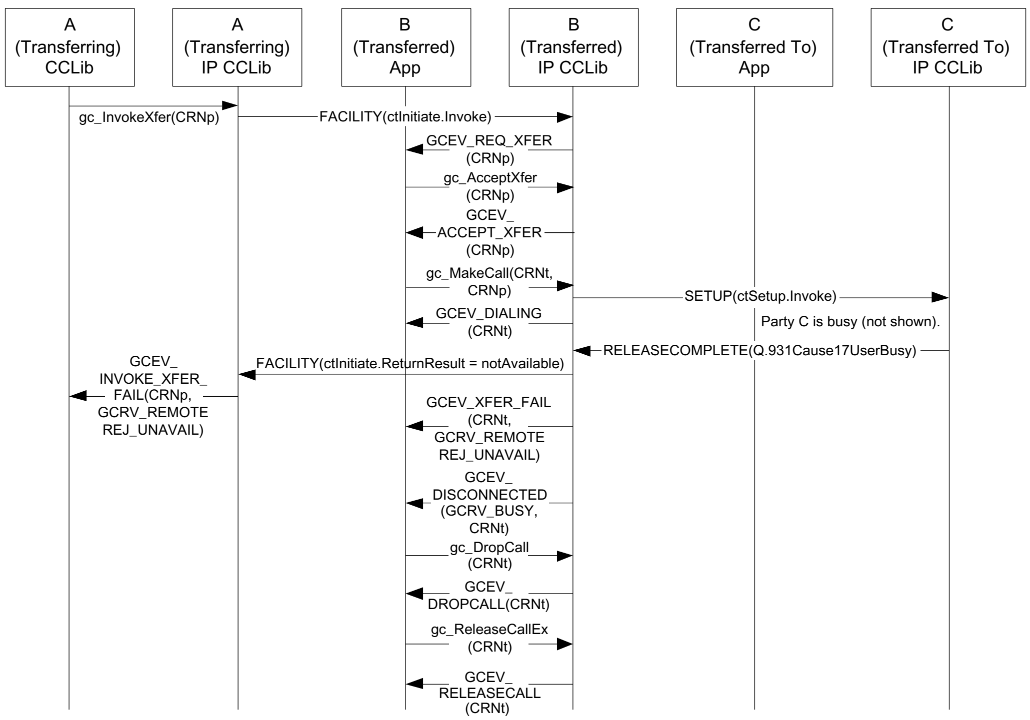


3.2.3.5 Party C is Busy When Transfer Attempted

The transferred-to endpoint (party C) may also be busy at the time of transfer (unknown to the transferring endpoint). As indicated in Figure 15, this would result in a Release Complete message with Q.931 Cause 17, User Busy, being returned to the transferred endpoint (party B) indicated to its application via a GCEV_DISCONNECTED event with a cause of GCRV_BUSY. The transferred endpoint (party B) returns a ctInitiate.ReturnError APDU to the transferring endpoint to indicate the transfer failure and in turn must drop the transferred call attempt. As a result, the GCEV_INVOKE_XFER_FAIL termination event is received at transferring endpoint (party A) and the original primary call is left connected and in the GCST_CONNECTED state from the perspective of both A and B.

Figure 15. Blind Call Transfer Failure - Party C is Busy When Transfer Attempted

Precondition: Primary call between A and B is in connected (not shown).
Party C has call connected to another party (not shown).



Post condition: Parties A and B remain connected. Party C also remains connected (to another party not shown).

3.2.4 Successful Supervised Call Transfer Scenario

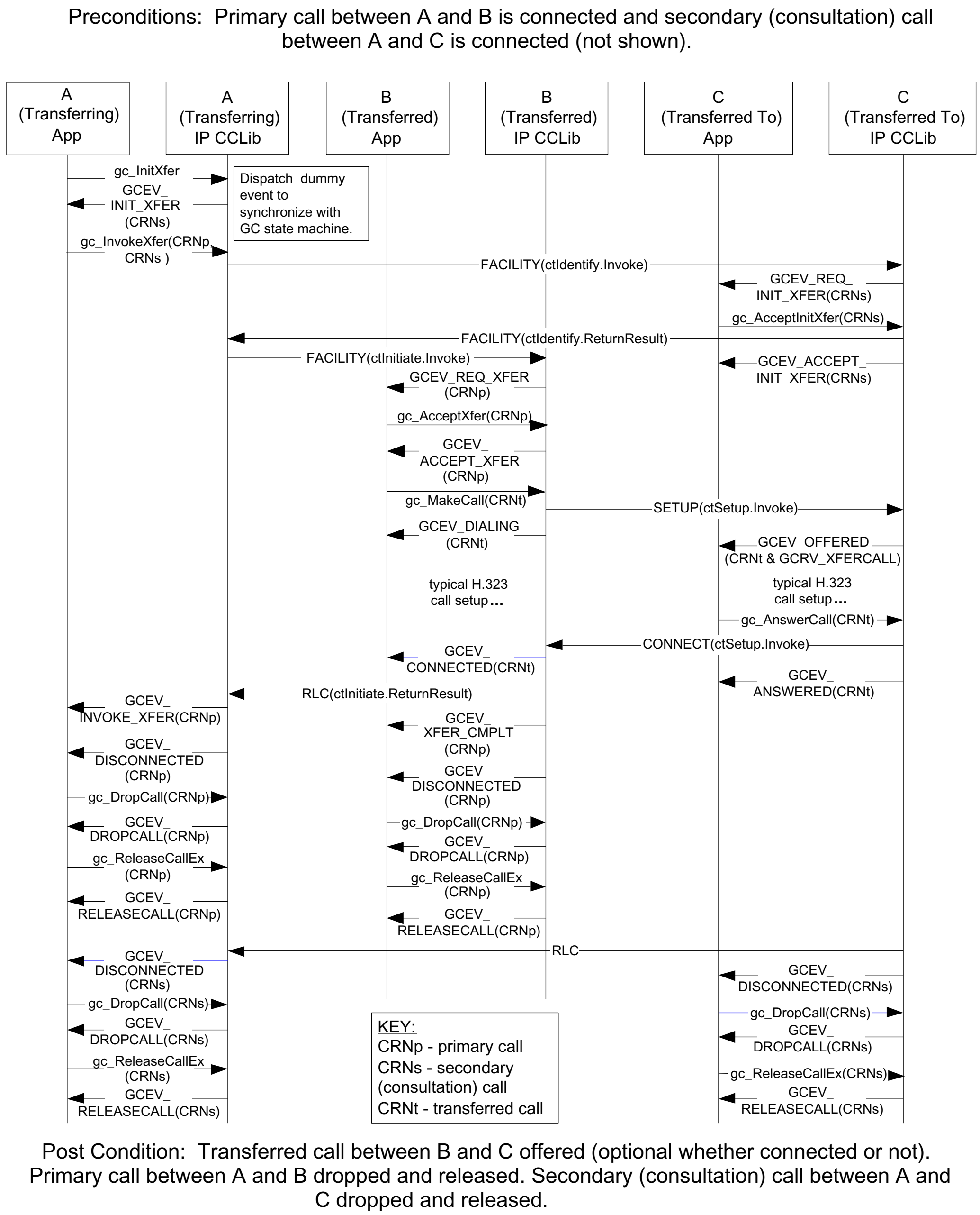
As indicated in Figure 16, the first precondition for supervised transfer is that the transferring endpoint (party A) and the transferred endpoint (party B) are participating in an active call (the primary call) and from the Global Call perspective, in the GCST_CONNECTED state.

The second precondition for supervised transfer is that a secondary call (the consultation call) from the transferring endpoint (party A) to the transferred-to endpoint (party C) is active and both endpoints are in the GCST_CONNECTED state.

Completion of a successful supervised transfer results in the eventual termination of the primary and secondary (consultation) calls, and the creation of the transferred call. Note that the connection of the transferred call is not a mandate for supervised call transfer. While less likely due to the typical human dialogue on a secondary (consultation) call, it is always possible that the transferred call itself may be left unanswered and eventually abandoned and still be considered a *successful* transfer from the signaling perspective of the transferring endpoint (party A).

For simplification purposes Figure 16 does not indicate the opening and closing of logical channels (and the associated media sessions) as the control procedures are consistent with typical non-transfer related H.323 calls.

Figure 16. Successful Supervised Call Transfer



Transferring Endpoint (Party A)

As in the case of blind transfer, the transferring endpoint initiates the supervised transfer by calling the **gc_InvokeXfer( )** function. The distinction between blind and supervised transfer usage is the addition of the CRN of the secondary (consultation) call. Calling the **gc_InvokeXfer( )** function at the transferring endpoint with two CRN values results in the sending of an ctIdentify.Invoke APDU in a FACILITY message to the transferred-to endpoint (party C). Once a positive acknowledgement from the transferred-to endpoint is received via a ctIdentify.ReturnResult APDU in a FACILITY message, the transferring endpoint automatically proceeds to invoke the actual call transfer by sending an ctInitiate.Invoke APDU in a FACILITY message to the transferred endpoint (party B).

From this point forward, from the perspective of this endpoint, the behavior is similar to that of a blind or unsupervised transfer. The one difference is that the secondary, consultation call is disconnected once the transferred call is answered at the transferred-to endpoint (party C) and must be explicitly dropped and released. Note however, if the transferred call goes unanswered or fails, the secondary call is left active and maintained in the GCST_CONNECTED state.

Transferred Endpoint (Party B)

The endpoint to be transferred (party B) has no knowledge of the origins of the destination address information a priori in that it was retrieved as a result of a consultation call. Thus, from the perspective of this endpoint, the behavior and handling of supervised transfer is exactly the same as that of blind transfer.

Transferred-To Endpoint (Party C)

At any point in time after a secondary consultation call is answered by the transferred-to endpoint, a FACILITY(ctIdentify.Invoke) request may arrive requesting whether it is able to participate in an upcoming transfer as signified by the GCEV_REQ_INIT_XFER event. Assuming that the endpoint is capable, the application calls the **gc_AcceptInitXfer( )** function to accept the transfer along with the intended rerouting number address in the **reroutinginfop** GC_REROUTING_INFO pointer parameter. The IP CCLIB internally returns a newly created callIdentity for the transferred call to use.

From this point forward, the behavior in handling the incoming transferred call from the perspective of this endpoint is identical to that of a blind or unsupervised transfer. The only difference is that the secondary, consultation call is disconnected once the transferred call is answered and must be explicitly dropped and released.

3.2.5 Unsuccessful Supervised Transfer Scenarios

Note: The same failures that can potentially occur in blind transfer, may take place in supervised transfer as well. See Section 3.2.3, "Unsuccessful Blind Transfer Scenarios", on page 45 for more information. The difference being that the secondary, consultation may optionally be cleared as specified in H.450.2.

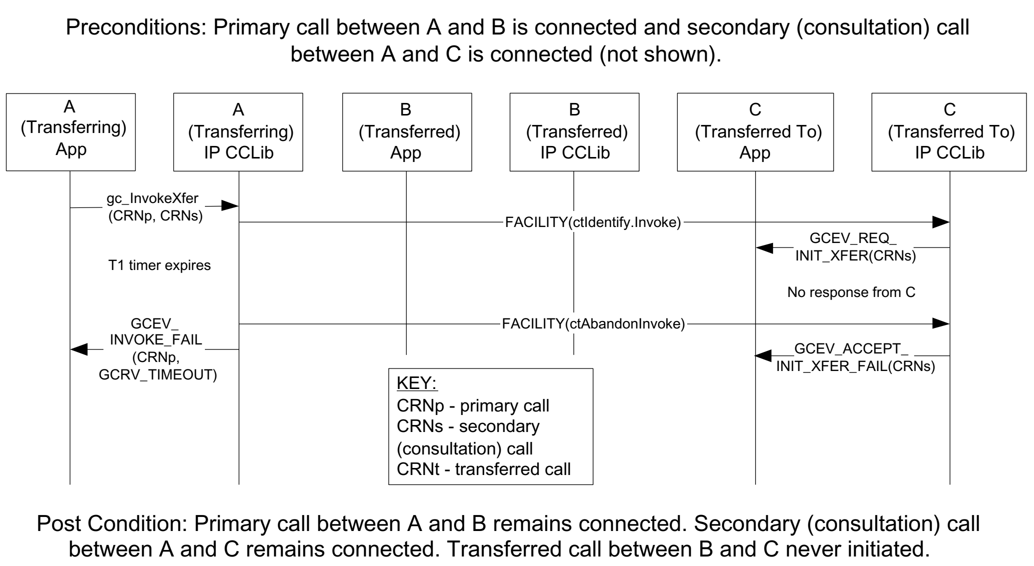
There are a several other scenarios where a supervised call transfer may fail. The most common scenarios are described in the following topics:

- Party C Timeout
- Party C Rejects the Transfer Request
- Party B Rejects the Transfer Request
- Party B Timeout

3.2.5.1 Party C Timeout

As indicated in Figure 17, the user or application at the transferred-to endpoint (party C) may fail to respond to the ctIdentifiy.Invoke request causing the timer 1 to expire at the transferring endpoint (party A) resulting in an abandoned transfer attempt. As a result, the GCEV_INVOKE_XFER_FAIL termination event is received at transferring endpoint (party A). Both the original primary call and the secondary, consultation call are left connected and in the GCST_CONNECTED state from the perspective of both A and B (primary) and A and C (secondary).

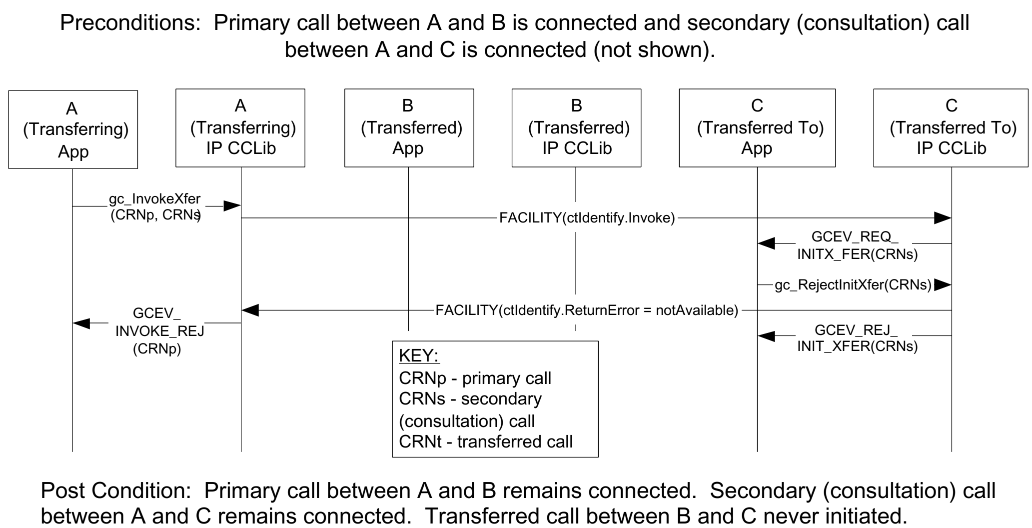
Figure 17. Supervised Call Transfer Failure - Party C Timeout



3.2.5.2 Party C Rejects the Transfer Request

As indicated in Figure 17, the user or application at the transferred-to endpoint (party C) may call the **gc_RejectInitXfer( )** function to signal via the ctInIdentify.ReturnResult APDU that it cannot participate in a transfer. As a result, the GCEV_INVOKE_XFER_REJ termination event is received at transferring endpoint (party A). Both the original primary call and the secondary, consultation call are left connected and in the GCST_CONNECTED state from the perspective of both A and B (primary) and A and C (secondary); GCST_CONNECTED state from the perspective of both A and B.

Figure 18. Supervised Call Transfer Failure - Party C Rejects the Transfer Request

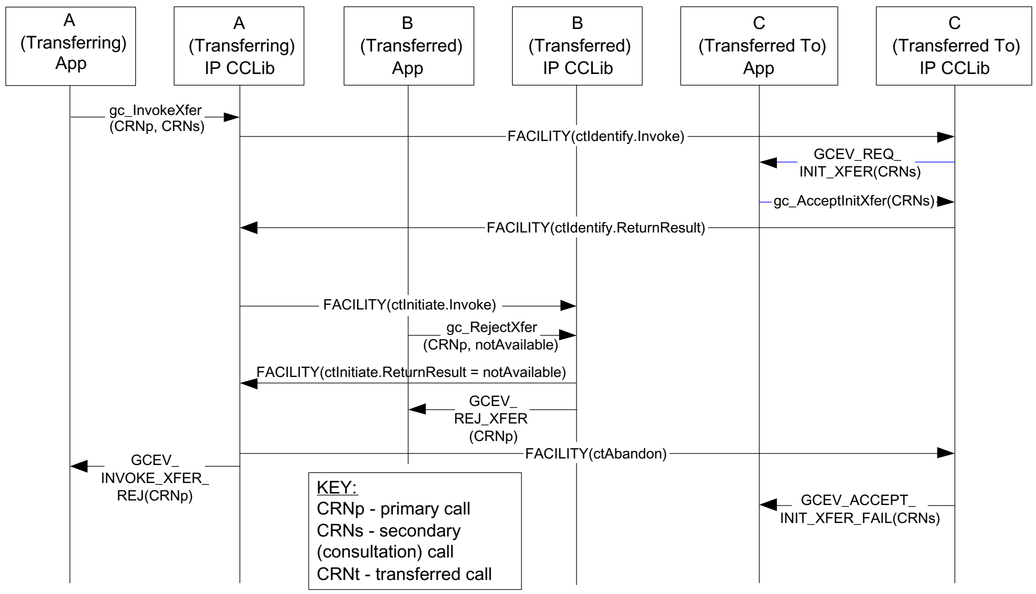


3.2.5.3 Party B Rejects the Transfer Request

As indicated in Figure 19, the user or application at the transferred endpoint (party B) may call the **gc_RejectXfer( )** function to reject the transfer request and signal via the ctInitiate.ReturnResult APDU that it cannot participate in a transfer. As a result, the GCEV_INVOKE_XFER_REJ termination event is received at transferring endpoint (party A). Both the original primary call and the secondary, consultation call are left connected and in the GCST_CONNECTED state from the perspective of both A and B (primary) and A and C (secondary); GCST_CONNECTED state from the perspective of both A and B.

Figure 19. Supervised Call Transfer Failure - Party B Rejects the Transfer Request

Preconditions: Primary call between A and B is connected and secondary (consultation) call between A and C is connected (not shown).



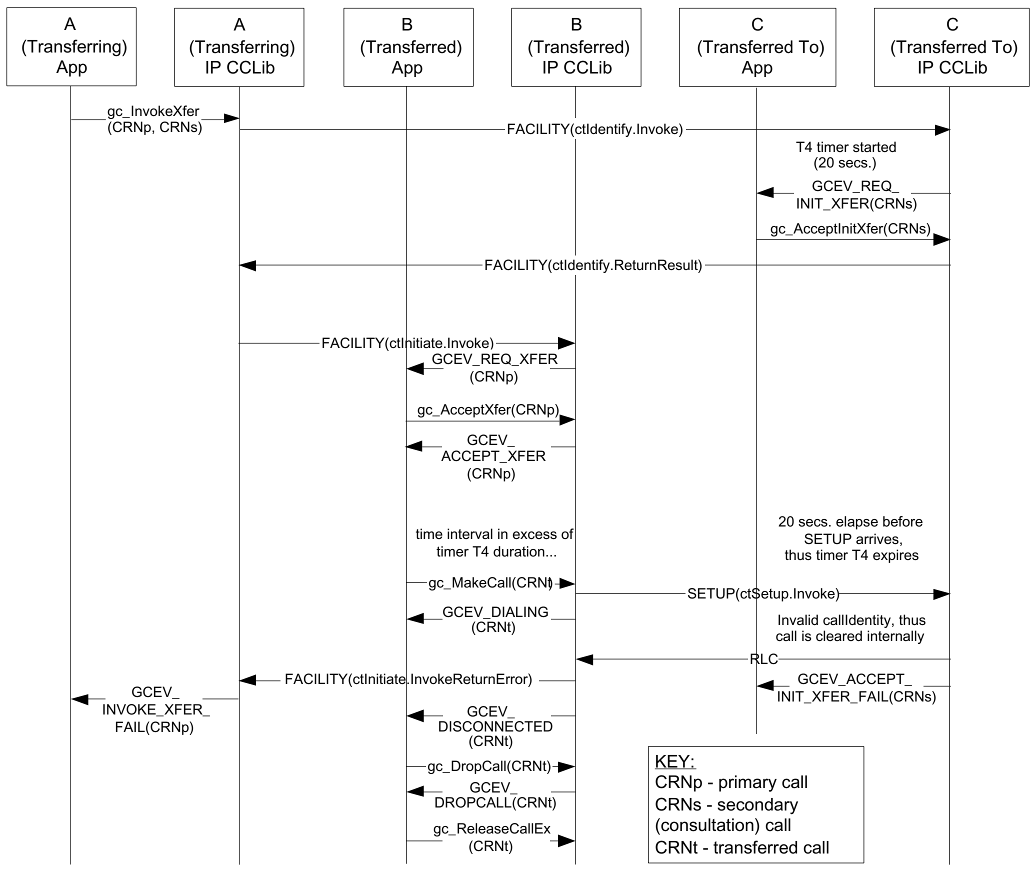
Post Condition: Primary call between A and B remains connected. Secondary (consultation) call between A and C remains connected.

3.2.5.4 Party B Timeout

As indicated in Figure 20, the user or application at the transferred-to endpoint (party C) may receive the transferred call after the T4 timer expires. If this is the case and the callIdentity is cleared as a result of the T4 expiry, the transferred-to endpoint will clear or reject the transferred call as indicated by a GCEV_DISCONNECTED event at the transferred endpoint (party B) and a GCEV_INVOKEXFER_FAIL event at the transferring endpoint (party A). Both the original primary call and the secondary, consultation call are left connected and in the GCST_CONNECTED state from the perspective of both A and B (primary) and A and C (secondary); GCST_CONNECTED state from the perspective of both A and B.

Figure 20. Supervised Call Transfer Failure - Party B Timeout

Preconditions: Primary call between A and B is connected and secondary (consultation) call between A and C is connected (not shown).



Post Condition: Primary call between A and B remains connected. Secondary (consultation) call between A and C remains connected.

3.3 T.38 Fax Server Call Scenarios

Global Call supports T.38 fax server as described in Section 4.18, “T.38 Fax Server Support”, on page 126. The following scenarios demonstrate the T.38 fax server capabilities provided when using IP technology (both H.323 and SIP):

- Sending T.38 Fax in an Established Audio Session
- Receiving T.38 Fax in an Established Audio Session
- Sending T.38 Fax Without an Established Audio Session
- Receiving T.38 Fax Without an Established Audio Session
- Sending a Request to Switch From T.38 Fax to Audio
- Receiving a Request to Switch From T.38 Fax to Audio

- Terminating a Call After a T.38 Fax Session
- Recovering from a T.38 Fax to Audio or Vice Versa Switch Failure

Note: In these scenarios, the application must include T.38 Fax capability either when using **gc_MakeCall( )** for an outbound call or when using **gc_CallAck( )**, **gc_AcceptCall( )**, or **gc_AnswerCall( )** for an inbound call.

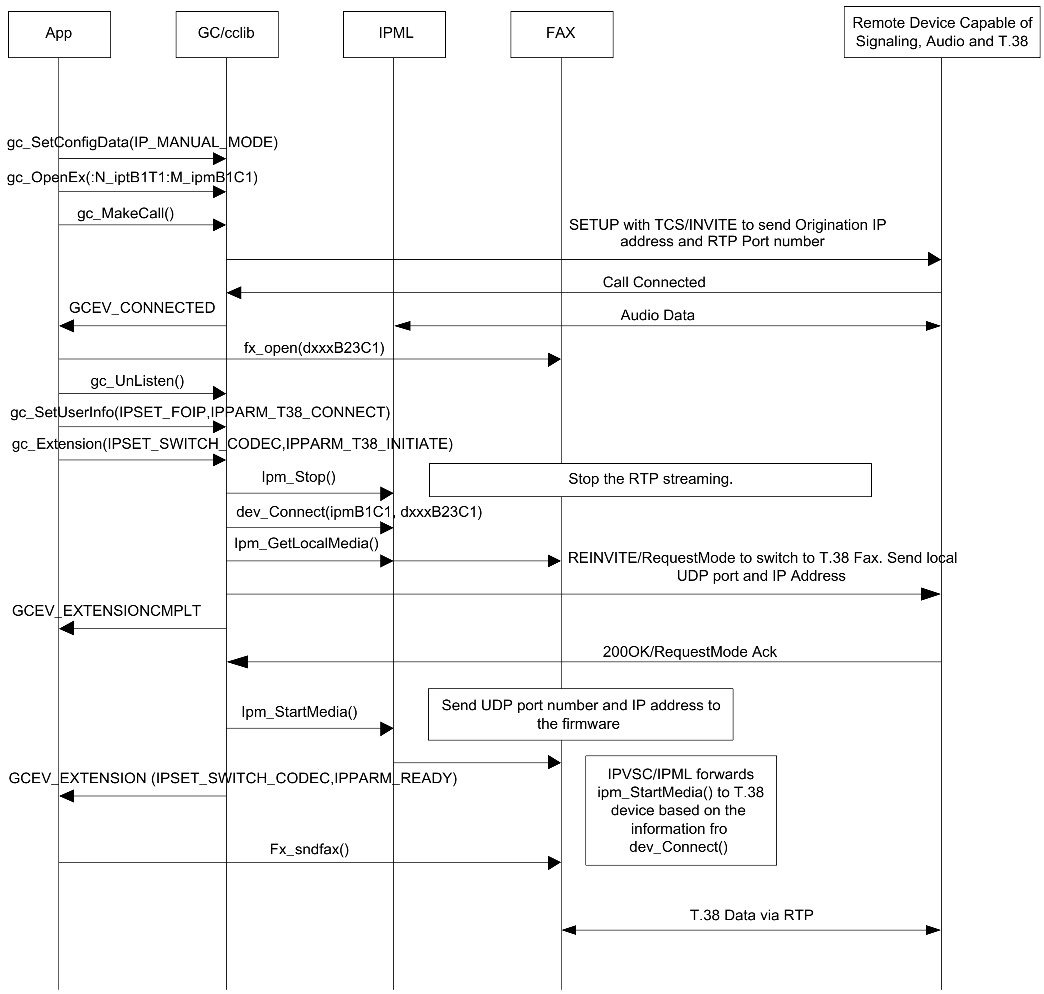
3.3.1 Sending T.38 Fax in an Established Audio Session

In this scenario, the user application uses the Global Call API to open a Media device, configures "Manual" mode of operation and establishes an audio session with the remote device. See Section 4.18.2, "Specifying Manual Operating Mode", on page 128 for more information on manual mode. A T.38 Fax device is then opened and the application switches from an audio session to a T.38 session.

When the application receives notification that the T.38 session is ready, fax information can be sent. Figure 21 shows the scenario diagram.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 21 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 21. Sending T.38 Fax in an Established Audio Session



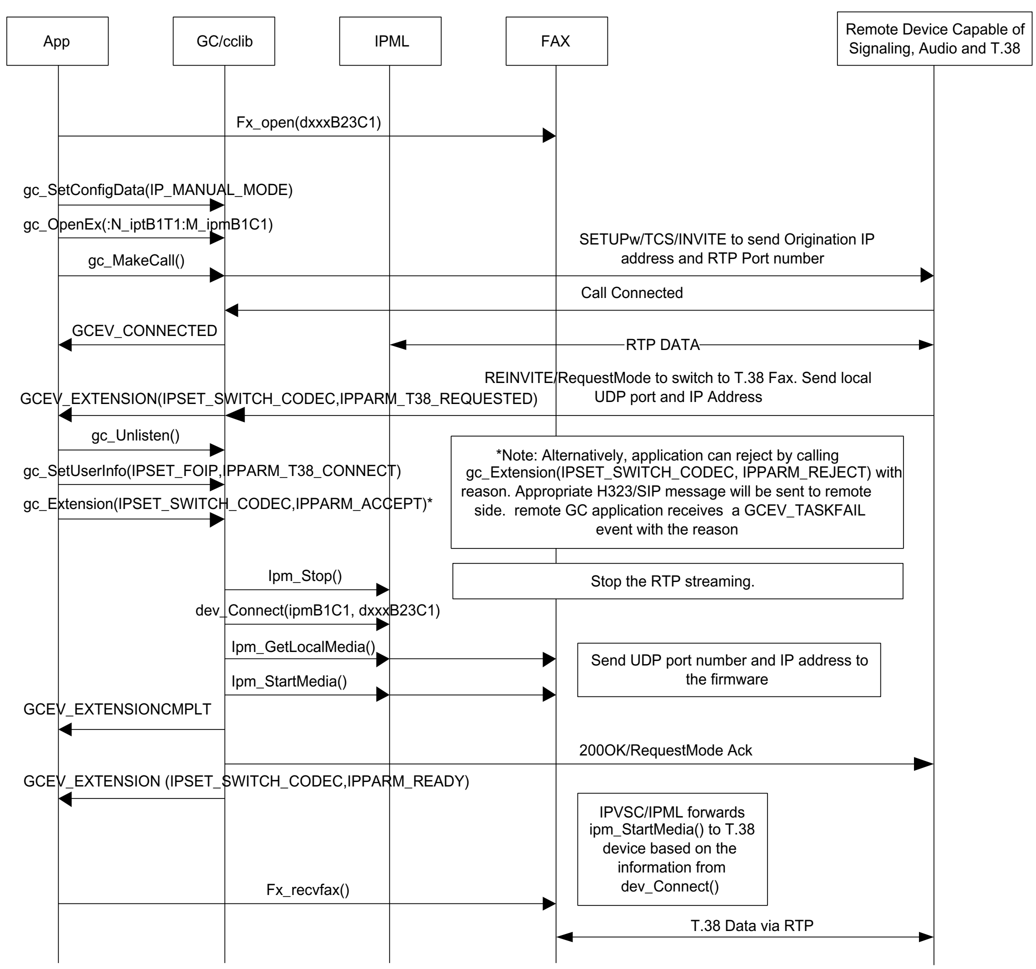
3.3.2 Receiving T.38 Fax in an Established Audio Session

In this scenario, the user application uses the Global Call API to open a Media device, configures "Manual" operating mode and establishes an audio session with the remote device. See Section 4.18.2, "Specifying Manual Operating Mode", on page 128 for more information on manual mode. To prepare to receive fax, the application must also open a T.30 Fax device. During the audio session, the application can be notified of an incoming request to switch from audio to T.38 fax.

The application can choose to accept or reject this request. If the user chooses to accept, Global Call notifies the application that the T.38 session is ready to receive a fax. Figure 22 shows the scenario diagram.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 22 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 22. Receiving T.38 Fax in an Established Audio Session



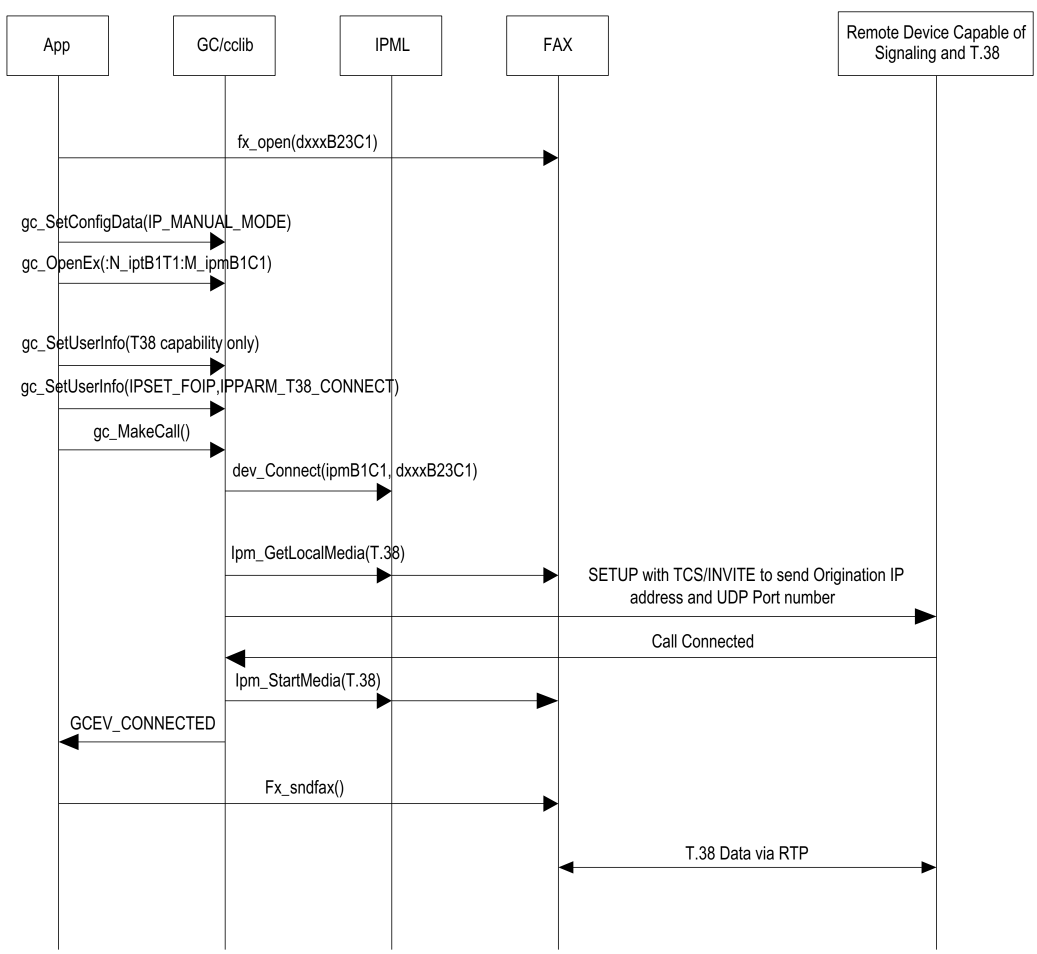
3.3.3 Sending T.38 Fax Without an Established Audio Session

This scenario describes the sending of T.38 Fax in a media session that does not have audio already established. The application first opens a Media device and a T.38 Fax device and configures "Manual" mode of operation. See Section 4.18.2, "Specifying Manual Operating Mode", on page 128 for more information on manual mode. The Global Call API is then used to associate the T.38 Fax device with the IP Media device before making a new T.38 call.

Once the call is connected, the application can send a fax. Figure 23 shows the scenario diagram.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 23 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 23. Sending T.38 Fax Without an Established Audio Session



3.3.4 Receiving T.38 Fax Without an Established Audio Session

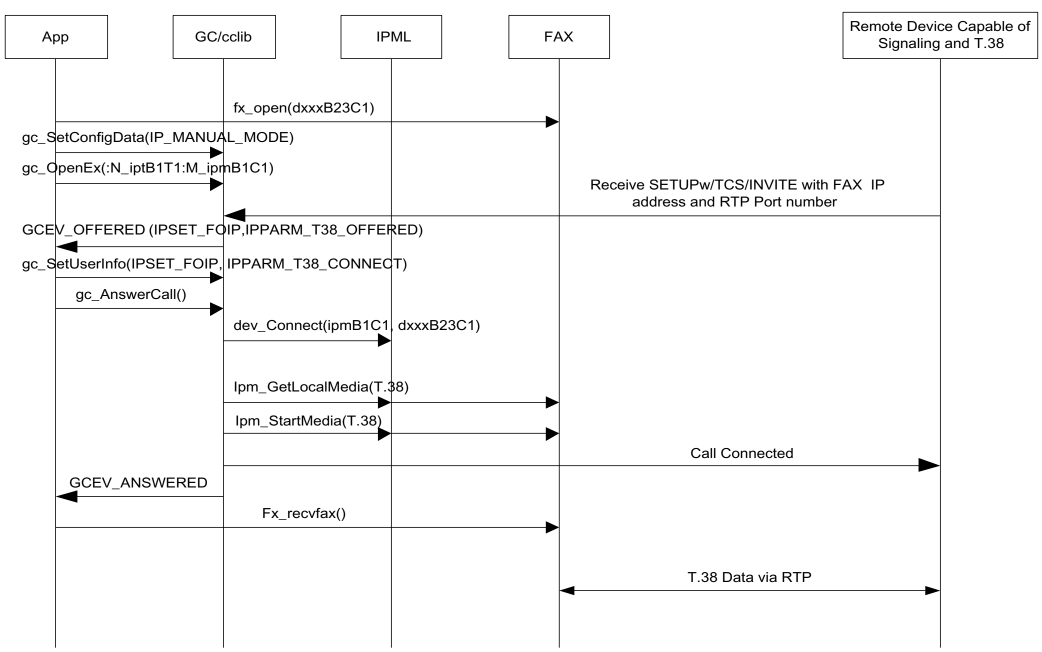
This scenario describes the reception of T.38 Fax in a media session that does not have audio already established. The application first opens a Media device and a T.38 Fax device and configures "Manual" operating mode. See Section 4.18.2, "Specifying Manual Operating Mode", on page 128 for more information on manual mode. When the application receives a T.38 fax request, a GCEV_OFFERED event with T.38 extension information is received.

If the application accepts the call, the T.38 Fax device is associated with the Media device before the T.38 call is answered. Figure 24 shows the scenario diagram.

Notes: **1.** The GCEV_OFFERED event with T.38 extension information is only generated if the following requirements are met. For H.323, the incoming message must be a Q.931 Setup message with data terminal capability only. For SIP, the incoming message must be an INVITE message with an SDP that has an image media descriptor only. If this condition is not met, the GCEV_OFFERED event does not include any T.38 extension information. This limitation prevents the T.38 server from receiving the T.38 request in H.323 slow start or in a SIP no SDP INVITE request.

2. The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 24 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 24. Receiving T.38 Fax Without an Established Audio Session

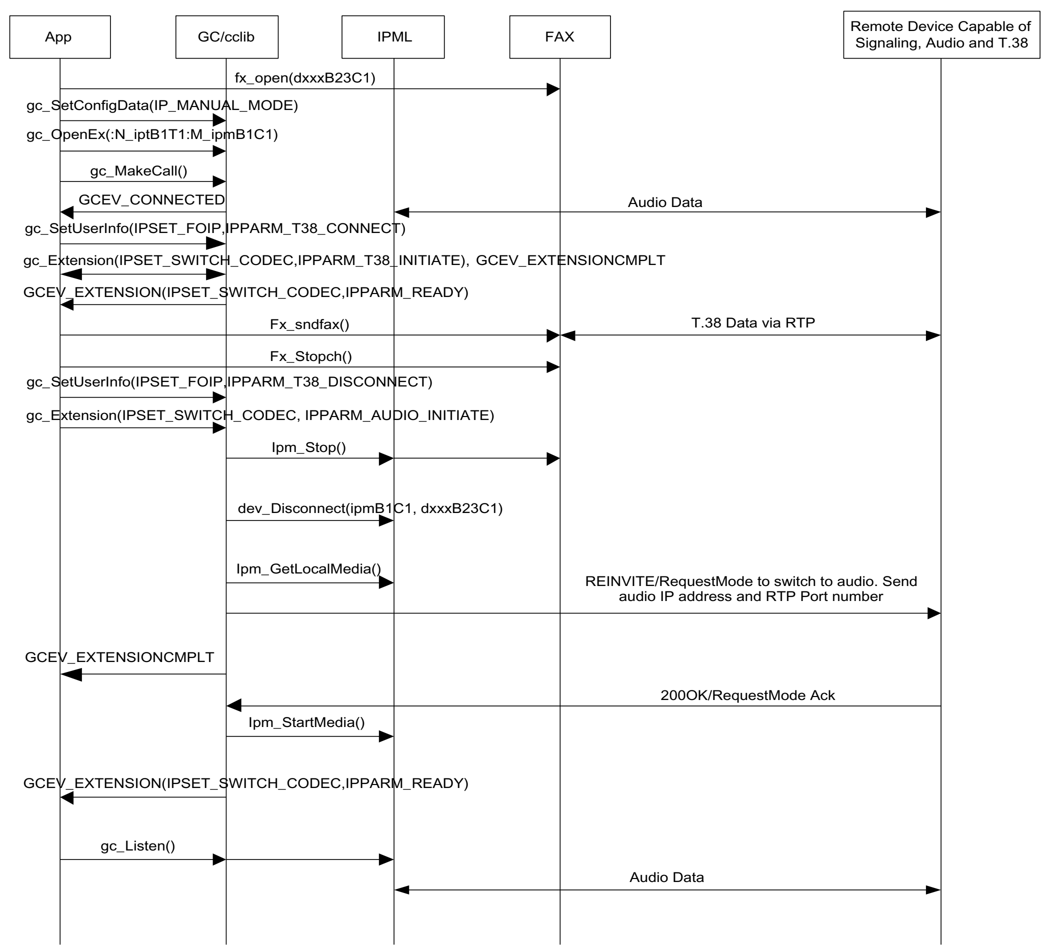


3.3.5 Sending a Request to Switch From T.38 Fax to Audio

In a fax session, when a fax completes, the application can use the Global Call API to issue a request to switch from a T.38 fax session back to an audio session after disassociating the T.38 Fax device from the Media device. When Global Call notifies the application that the audio session has been reestablished, the application can once again send and receive audio. Figure 25 shows the scenario diagram.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 25 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 25. Sending a Request to Switch From T.38 Fax to Audio

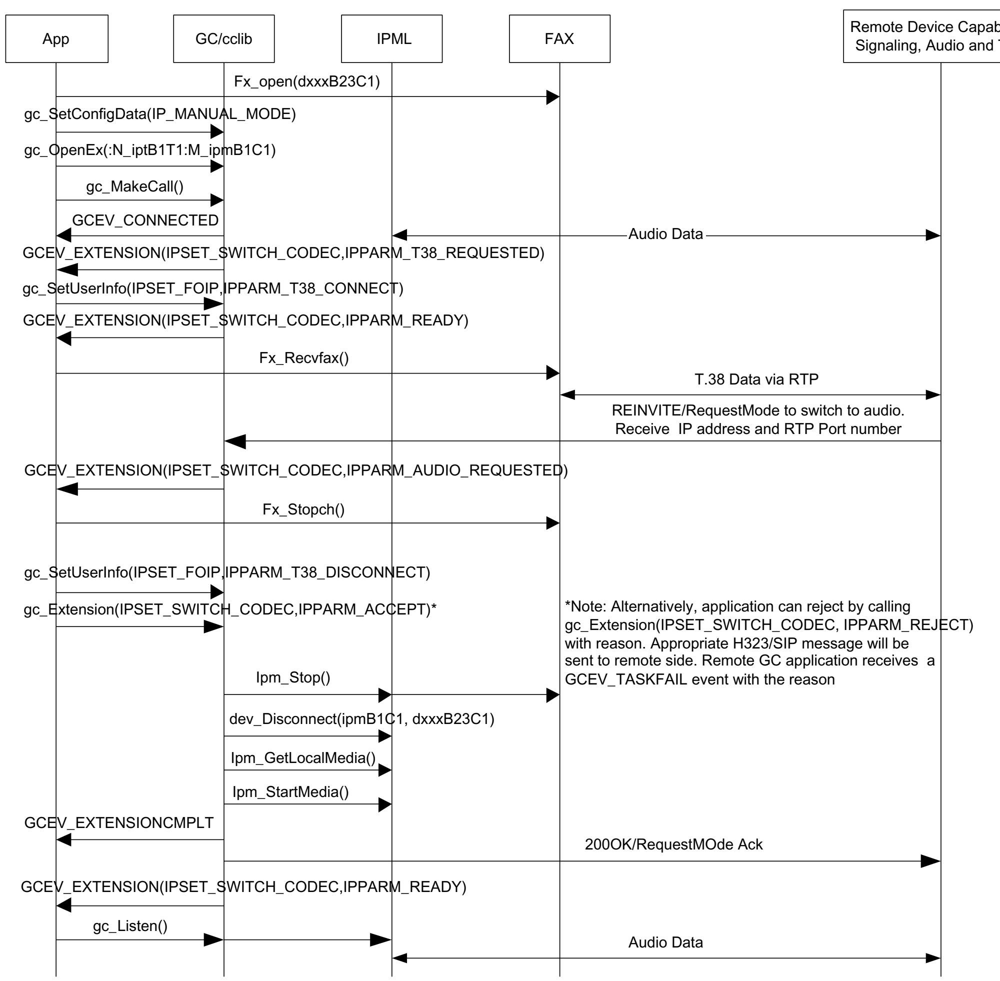


3.3.6 Receiving a Request to Switch From T.38 Fax to Audio

In a fax session, when a fax completes, the application can receive a request to switch from a T.38 fax session back to an audio session. The application can choose to accept the request after disassociating the T.38 Fax device from the Media device. When Global Call notifies the application that the audio session has been reestablished, the application can once again send and receive audio. Figure 26 shows the scenario diagram.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 26 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 26. Receiving a Request to Switch From T.38 Fax to Audio



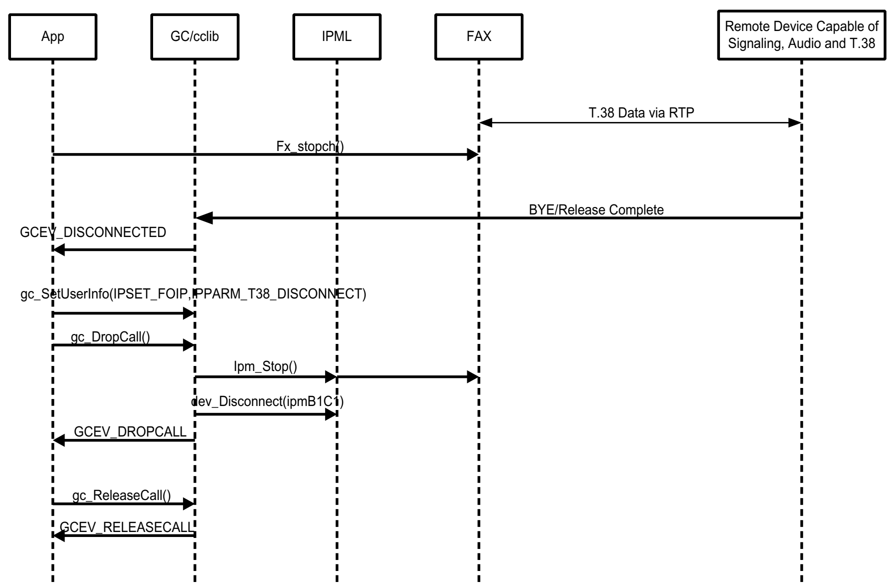
3.3.7 Terminating a Call After a T.38 Fax Session

In any scenario where a T.38 session is established and fax is complete, the application can terminate the call without switching to audio. In either outbound or inbound call termination, the application must disassociate the T.38 Fax device from the Media device before calling **gc_DropCall( )**. This ensures the Media device in the correct state for the next call.

Terminating a call after an audio session follows the normal Global Call call procedures.

Note: The application must not use both Global Call and IP Media Library functions on the same device. The IP Media Library calls (ipm_) in Figure 27 are shown for informational purposes only. Global Call interacts with the IP Media Library on behalf of the application.

Figure 27. Terminating a Call After a T.38 Fax Transfer.



3.3.8 Recovering from a T.38 Fax to Audio or Vice Versa Switch Failure

Switching to T.38 Fax or audio may fail due to any a number of reasons, for example, rejection or no response from remote endpoint. It is highly recommended that the application set up a timer for a minimum of 35 seconds for each switching request. If a timeout occurs while waiting for a GCEV_EXTENSION event that has an associated IPPARM_READY parameter, the application has two options:

- Attempt to switch back to original session as if the GCEV_EXTENSION event were received without media capability.
- Terminate the call as if GCEV_EXTENSION event were received without media capability.

If the application times out when switching to T.38 Fax (that is, it does not receive a GCEV_EXTENSION event with an IPPARM_READY parameter within the timeout period), it should follow the scenarios described in Section 3.3.5, “Sending a Request to Switch From T.38 Fax to Audio”, on page 61, Section 3.3.6, “Receiving a Request to Switch From T.38 Fax to Audio”, on page 62, or Section 3.3.7, “Terminating a Call After a T.38 Fax Session”, on page 63.

Note: The application must call the **gc_SetUserInfo( )** function with a GC_PARM_BLK that contains a set ID of IPSET_FOIP and a parameter ID of IPPARM_T38_DISCONNECT to disassociate the devices in any of the scenarios.

If the application times out when switching to audio (that is, it does not receive a GCEV_EXTENSION event with an IPPARM_READY parameter within the timeout period), it should follow the scenarios described in Section 3.3.1, “Sending T.38 Fax in an Established Audio Session”, on page 56, Section 3.3.2, “Receiving T.38 Fax in an Established Audio Session”, on page 58, or drop the call as if in audio session.

IP-Specific Operations 4

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

4.1 Call Control Library Initialization

Certain configuration parameters, such as the maximum number of IPT devices available, the local IP address, and the call signaling port, are configurable when using the **gc_Start( )** function to initialize Global Call. For example, the default maximum number of IPT devices is 120, but is configurable to as many as 2016.

Before using the **gc_Start( )** function, the **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** functions must be used to initialize the IPCCLIB_START_DATA and IP_VIRTBOARD data structures. These functions set default values that can then be overridden with desired values. In particular, it may be desirable to override the default values in the IP_VIRTBOARD structure in order to set the terminal type, to enable access to H.323 message information fields, to enable the call transfer supplementary service for H.450.2 protocol, or to enable access to SIP message information fields. See Section 7.2.26, "gc_Start( ) Variances for IP", on page 201, and the reference page for IP_VIRTBOARD on page 243 for more information.

Note: In the IPCCLIB_START_DATA structure, the maximum value of the num_boards field, which defines the number of NICs or NIC addresses, is 8.

4.2 Using Fast Start and Slow Start Setup

Fast start and slow start are supported in both the H.323 and SIP protocols. 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 endpoint.

In H.323, fast start and slow start setup are supported depending on the version of H.323 standard supported at the remote side. If the remote side supports H.323 version 2 or above, fast start setup can be used; otherwise, a slow start setup is used. Fast start connection reduces the time required to set up a call to one round-trip delay following the H.225 TCP connection. The concept is to include all the necessary parameters for the logical channel to be opened (H.245 information) in the Setup message. The logical channel information represents a set of supported capabilities from which the remote end can choose the most appropriate capability. If the remote side decides to use fast start connection, it returns the desired logical channel parameters in the Alerting, Proceeding, or Connect messages.

Note: In an H.323 fast start call, the fast start element (FSE) is included in outgoing H.225 PROCEEDING / ALERTING only when the user explicitly specifies the coders. If no coder is specified (either a preferred coder or "don't care") before **gc_CallAck( )** and **gc_AcceptCall( )** the FSE is not sent out until CONNECT (i.e., after **gc_AnswerCall( )**).

In SIP, fast start and slow start setup are also supported. In slow start setup, the INVITE message will have no Session Description Protocol (SDP) and therefore the remote side will propose the session attributes in the SDP of the ACK message.

In Global Call, fast start and slow start connection are supported on a call-by-call basis. Fast start connection is used by default, but slow start connection can be forced by including the IPPARM_CONNECTIONMETHOD parameter ID with a value of IP_CONNECTIONMETHOD_SLOWSTART in the ext_datap field (of type GC_PARM_BLK) in the GCLIB_MAKECALL_BLK structure associated with a call. The following code segment shows how to specify a slow start connection explicitly by including the IPPARM_CONNECTIONMETHOD parameter ID when populating the ext_datap field:

```
gc_util_insert_parm_val(&libBblock.ext_datap, IPSET_CALLINFO, IPPARM_CONNECTIONMETHOD,
         sizeof(char), IP_CONNECTIONMETHOD_SLOWSTART);
```

In addition, the IPPARM_CONNECTIONMETHOD parameter ID can be set to a value of IP_CONNECTIONMETHOD_FASTSTART to force a fast start connection on a line device configured to use a slow start connection (using **gc_SetUserInfo( )** with a **duration** parameter of GC_ALLCALLS).

Note: In SIP, only the calling side can choose fast start or slow start, unlike H.323 where both sides can select either fast start or slow start.

4.3 Setting Call-Related Information

Global Call allows applications to set many items of call-related information. The following topics are presented in this section:

- Overview of Setting Call-Related Information
- Setting Coder Information
- Specifying Nonstandard Data Information When Using H.323
- Specifying Nonstandard Control Information When Using H.323
- Setting and Retrieving Disconnect Cause or Reason Values
- Setting Busy Reason Codes

4.3.1 Overview of Setting Call-Related Information

Table 1 summarizes the types of information elements that can be specified, the corresponding set IDs and parameter IDs used to set the information, the functions that can be used to set the information, and an indication of whether the information is supported when using H.323, SIP, or both. For more information on the various parameters, refer to the corresponding parameter set reference section in Chapter 8, "IP-Specific Parameters".

Table 1. Summary of Call-Related Information that can be Set

Type of Information	Set ID and Parameter IDs	Functions Used to Set Information	SIP/ H.323
Bearer Capability IE	IPSET_CALLINFO • IPPARM_BEARERCAP	**gc_SetUserInfo( )** (GC_SINGLECALL only)	H.323 only
Call ID (GUID)	IPSET_CALLINFO • IPPARM_CALLID **Note:** Setting the Call ID must be done judiciously because it might affect the call control implementation supported by the stack. The Call ID should be treated as a GUID and should be unique at all times.	**gc_SetUserInfo( )** (GC_SINGLECALL only) **gc_MakeCall( )**	both
Coder Information †	GCSET_CHAN_CAPABILITY • IPPARM_LOCAL_CAPABILITY	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	both
Conference Goal	IPSET_CONFERENCE • IPPARM_CONFERENCE_GOAL	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
Connection Method	IPSET_CALLINFO • IPPARM_CONNECTIONMETHOD	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	both

† If no transmit or receive coder type is specified, any supported coder type is accepted. The default is "don't care"; that is, any media coder supported by the platform is valid.
†† The duration parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
††† On the terminating side, can only be set using **gc_SetConfigData( )** on a board device. See Section 4.13, "Enabling and Disabling Tunneling in H.323", on page 107 for more information.

Table 1. Summary of Call-Related Information that can be Set (Continued)

Type of Information	Set ID and Parameter IDs	Functions Used to Set Information	SIP/ H.323
DTMF Support	IPSET_DTMF • IPPARM_SUPPORT_DTMF_BITMASK	**gc_SetConfigData( )** **gc_SetUserInfo( )** ††	both
Display Information	IPSET_CALLINFO • IPPARM_DISPLAY	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	both
Enabling/Disabling Unsolicited Events	IPSET_EXTENSIONEVT_MSK • GCACT_ADDMSK • GCACT_SETMSK • GCACT_SUBMSK	**gc_SetConfigData( )**	both
Facility IE	IPSET_CALLINFO • IPPARM_FACILITY	**gc_SetUserInfo( )** (GC_SINGLECALL only)	H.323 only
MediaWaitFor Connect	IPSET_CALLINFO • IPPARM_MEDIAWAITFORCONNECT	**gc_SetUserInfo( )** (GC_SINGLECALL only) **gc_MakeCall( )**	H.323 only
Nonstandard Control Information	IPSET_NONSTANDARDCONTROL Either: • IPPARM_NONSTANDARDDATA_DATA and IPPARM_NONSTANDARDDATA_OBJID or • IPPARM_NONSTANDARDDATA_DATA and IPPARM_H221NONSTANDARD	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
Nonstandard Data	IPSET_NONSTANDARDDATA Either: • IPPARM_NONSTANDARDDATA_DATA and IPPARM_NONSTANDARDDATA_OBJID or • IPPARM_NONSTANDARDDATA_DATA and IPPARM_H221NONSTANDARD	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
Phone List	IPSET_CALLINFO • IPPARM_PHONELIST	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	both
Presentation Indicator	IPSET_CALLINFO • IPPARM_PRESENTATION_IND	**gc_SetUserInfo( )** (GC_SINGLECALL only) **gc_MakeCall( )**	H.323 only

† If no transmit or receive coder type is specified, any supported coder type is accepted. The default is “don’t care”; that is, any media coder supported by the platform is valid.
†† The duration parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
††† On the terminating side, can only be set using **gc_SetConfigData( )** on a board device. See Section 4.13, “Enabling and Disabling Tunneling in H.323”, on page 107 for more information.

Table 1. Summary of Call-Related Information that can be Set (Continued)

Type of Information	Set ID and Parameter IDs	Functions Used to Set Information	SIP/ H.323
SIP Message Information Fields	IPSET_SIP_MSGINFO • IPPARM_CALLID_HDR • IPPARM_CONTACT_DISPLAY • IPPARM_CONTACT_URI • IPPARM_DIVERSION_URI • IPPARM_FROM_DISPLAY • IPPARM_REFERRED_BY • IPPARM_REPLACES • IPPARM_REQUEST_URI • IPPARM_TO_DISPLAY	**gc_SetUserInfo( )** (GC_SINGLECALL only)	SIP only
T.38 Fax device association or disassociation with Media device	IPSET_FOIP • IPPARM_T38_CONNECT • IPPARM_T38_DISCONNECT	**gc_SetUserInfo( )** †	both
Tunnelling†††	IPSET_CALLINFO • IPPARM_H245TUNNELING	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
Type of Service (ToS)	IPSET_CONFIG • IPPARM_CONFIG_TOS	**gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
User to User Information	IPSET_CALLINFO • IPPARM_USERUSER_INFO	**gc_SetConfigData( )** **gc_SetUserInfo( )** †† **gc_MakeCall( )**	H.323 only
Vendor Information	IPSET_VENDORINFO • IPPARM_H221NONSTD • IPPARM_VENDOR_PRODUCT_ID • IPPARM_VENDOR_VERSION_ID	**gc_SetConfigData( )**	H.323 only

† If no transmit or receive coder type is specified, any supported coder type is accepted. The default is “don’t care”; that is, any media coder supported by the platform is valid.
†† The duration parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
††† On the terminating side, can only be set using **gc_SetConfigData( )** on a board device. See Section 4.13, “Enabling and Disabling Tunneling in H.323”, on page 107 for more information.

4.3.1.1 Setting Call Parameters on a System-Wide Basis

Use the **gc_SetConfigData( )** function to configure call-related parameters including coder information. The values set by the **gc_SetConfigData( )** function are used by the call control library as default values for each line device.

See Section 7.2.24, “gc_SetConfigData( ) Variances for IP”, on page 195 for more information about the values of function parameters to set in this context.

4.3.1.2 Setting Call Parameters on a Per Line Device Basis

The **gc_SetUserInfo( )** function (with the **duration** parameter set to GC_ALLCALLS) can be used to set the values of call-related parameters on a per line-device basis. 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 7.2.25, "gc_SetUserInfo( ) Variances for IP", on page 198 for more information about the values of function parameters to set in this context.

4.3.1.3 Setting Call Parameters on a Per Call Basis

There are two ways to set call parameters on a per-call basis:

- Using **gc_SetUserInfo( )** with the **duration** parameter set to GC_SINGLECALL
- Using **gc_MakeCall( )**

Setting Per-Call Call Parameters Using gc_SetUserInfo( )

The **gc_SetUserInfo( )** function (with the **duration** parameter set to GC_SINGLECALL) can be used to set call parameter 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 7.2.25, "gc_SetUserInfo( ) Variances for IP", on page 198 for more information about the values of function parameters to set in this context.

Setting Per-Call Call Parameters Using gc_MakeCall( )

The **gc_MakeCall( )** function can be used to set call parameter values 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 7.2.16, "gc_MakeCall( ) Variances for IP", on page 172 for more information about the values of function parameters to set in this context.

4.3.2 Setting Coder Information

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

Coder information can be set in the following ways:

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

In each case, a GC_PARM_BLK is set up to contain the coder information. The GC_PARM_BLK must contain the GCSET_CHAN_CAPABILITY parameter set ID with the IPPARM_LOCAL_CAPABILITY parameter ID, which is of type IP_CAPABILITY.

Possible values for fields in the IP_CAPABILITY structure are:

- capability - Specifies the coder type from among the types supported by the particularIP telephony platform; see Table 2 for platform-specific coder types. The following values are defined for the capability field:
  - GCCAP_AUDIO_g711Alaw64k
  - GCCAP_AUDIO_g711Ulaw64k
  - GCCAP_AUDIO_g729AnnexA
  - GCCAP_AUDIO_g729AnnexAwAnnexB
  - GCCAP_AUDIO_NO_AUDIO
  - GCCAP_DATA_t38UDPFax
  - GCCAP_dontCare - The complete list of coders supported by a product is used when negotiating the coder type to be used. If multiple variations of the same coder are supported by a product, the underlying call control library offers the preferred variant only. For example, if G.711 10ms, 20ms, and 30ms are supported, only the preferred variant G.711, 20 ms is included.
- type - One of the following:
  - GCCAPTYPE_AUDIO
  - GCCAPTYPE_RDATA
- direction - One of the following:
  - IP_CAP_DIR_LCLTRANSMIT - transmit capability
  - IP_CAP_DIR_LCLRECEIVE - receive capability
  - IP_CAP_DIR_LCLRXTX - transmit and receive capability (T.38 only)

  Note: It is recommended to specify both the transmit and receive capabilities.
- payload_type - Not supported. The currently supported coders have static (pre-assigned) payload types defined by standards.
- extra - Reference to a data structure of type IP_AUDIO_CAPABILITY, which contains the following two fields:
  - frames_per_packet - The number of frames per packet.

  Note: For G.711 coders, the extra.frames_per_packet field is the frame size (in ms).
  - VAD - Enables or disables VAD; Values: GCPV_DISABLE, GCPV_ENABLE, GCCAP_dontCare

  Note: Applications must explicitly set this field to GCPV_ENABLE for the coders that implicitly support only VAD, such as GCCAP_AUDIO_g729AnnexAwAnnexB.

See the reference page for IP_CAPABILITY on page 235 for more information.

Table 2 shows the coders that are supported when using the Global Call API with Intel NetStructure Host Media Processing (HMP) software.

Table 2. Coders Supported for Host Media Processing (HMP)

Coder and Rate	Global Call # Define	Frames Per Packet (fpp) and Frame Size (ms)	VAD Support
G.711 A-law	GCCAP_AUDIO_g711Alaw64k	Frame Size[1]: 10, 20, or 30 ms (Frames Per Packet: fixed at 1 fpp)	Not supported; must be explicitly disabled
G.711 mu-law	GCCAP_AUDIO_g711Ulaw64k	Frame Size[1]: 10, 20, or 30 ms (Frames Per Packet: fixed at 1 fpp)	Not supported; must be explicitly disabled
G.723.1 5.3 kbps	GCCAP_AUDIO_g7231_5_3k	Frames Per Packet: 2 or 3 (Frame Size: fixed at 30 ms)	Supported
G.723.1, 6.3 kbps	GCCAP_AUDIO_g7231_6_3k	Frames Per Packet: 2 or 3 (Frame Size: fixed at 30 ms)	Supported
G.729a	GCCAP_AUDIO_g729AnnexA	Frames Per Packet: 2, 3, or 4 (Frame Size: fixed at 10 ms)	Not supported; must be explicitly disabled
G.729a+b	GCCAP_AUDIO_g729AnnexA wAnnexB	Frames Per Packet: 2, 3, or 4 (Frame Size: fixed at 10 ms)	Must be enabled [2]
T.38	GCCAP_DATA_t38UDPFax	Not applicable	Not applicable

Note:
1. 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 the reference page for IP_AUDIO_CAPABILITY on page 234 for more information.
2. Applications must explicitly specify VAD support even though G.729a+b implicitly supports VAD.

4.3.2.1 Specifying Media Capabilities Before Connection

Applications can only specify media capabilities before initial call connection. For an outbound call, capabilities must be set before or with the **gc_MakeCall( )**. For inbound calls, capabilities must be set before or with the **gc_AnswerCall( )**. Capability types can be GCCAPTYPE_AUDIO and/or GCCAPTYPE_RDATA. The result capabilities set by applications are listed in the Table 3.

Table 3. Capabilities Set by Application

GCCAPTYPE_AUDIO capability set by application	GCCAPTYPE_RDATA capability set by application	Result Capability
Not set	Not set	Support all audio capabilities during initial call connection. Support only switching between audio and T.38 during call connection. Do not support switching between different audio capabilities during call connection.
One or more GCCAP_AUDIO_XXX	Not Set	Support only audio capabilities specified no T.38 capability. Do not support switching between different audio capabilities during call connection.
Not Set	GCCAP_ DATA_t38UDPFax	Support only T.38 fax capability, no audio capability.

Table 3. Capabilities Set by Application (Continued)

GCCAPTYPE_AUDIO capability set by application	GCCAPTYPE_RDATA capability set by application	Result Capability
One or more GCCAP_AUDIO_XXX	GCCAP_ DATA_t38UDPFax	Support only audio capabilities specified during initial call connection. Support only switching between audio and T.38 during call connection. Do not support switching between different audio capabilities.
GCCAP_dontCare	Not Set	Support all audio capabilities during initial call connection. Do not support switching between different audio capabilities during call connection. Do not support T.38.
GCCAP_dontCare	GCCAP_ DATA_t38UDPFax	Support all audio capabilities during initial call connection. Support only switching between audio and T.38 during call connection. Do not support switching between different audio capabilities during call connection.

4.3.2.2 Resource Allocation When Using Low-Bit Rate Coders

The number of resources available when using G.723 and G.729 coders is limited. When all resources are consumed, depending on the requirements of the application, different behavior may be observed as follows:

- If the application specifies only G.723 and/or G.729 audio coders before **gc_MakeCall( )**, **gc_CallAck( )**, **gc_AcceptCall( )**, or **gc_AnswerCall( )**, the result is a function failure with an error code of IPERR_TXRXRESOURCESINSUFF.
- If the application specifies G.711 with G.723 and/or G.729 audio coders, only the G.711 coder will be provided in the capability set sent to the remote endpoint.
- If the application does not explicitly specify any audio capability, then the G.711 (both A-law and u-law) coders are included in the capability set sent to the remote endpoint.

The following applies to resource release:

- When a resource is not negotiated, the resource is released, permitting other channels to use the resource.
- When a resource is negotiated, using **gc_ReleaseCallEx( )** releases the resource.

4.3.3 Specifying Nonstandard Data Information When Using H.323

To specify Nonstandard Data, set up the GC_PARM_BLK pointed by the **infoparmblkp** function parameter with the IPSET_NONSTANDARDDATA parameter set ID, and one of two possible combinations of parameter IDs. First set the IPPARM_NONSTANDARDDATA_DATA parameter ID (maximum length MAX_NS_PARM_DATA_LENGTH or128). Then set either of the following two parameter IDs, depending on the type of object identifier to use:

- IPPARM_NONSTANDARDDATA_OBJID. The maximum length is MAX_NS_PARM_OBJID_LENGTH (40).

- IPPARM_H221NONSTANDARD

See Section 8.2.16, “IPSET_NONSTANDARDDATA”, on page 226 for more information.

The following code example shown how to set nonstandard data elements:

```
IP_H221NONSTANDARD appH221NonStd;
appH221NonStd.country_code = 181;
appH221NonStd.extension = 31;
appH221NonStd.manufacturer_code = 11;
char* pData = "Data String";
char* pOid = "1 22 333 4444";
choiceOfNSData = 1;/* App decides which type of object identifier to use */

/* setting NS Data */
gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_NONSTANDARDDATA,
                        IPPARM_NONSTANDARDDATA_DATA,
                        (unsigned char)(strlen(pData)+1),
                        pData);

    if (choiceOfNSData) /* App decides the CHOICE of OBJECTIDENTIFIER.
                           It cannot set both objid & H221 */
    {
        gc_util_insert_parm_ref(&pParmBlock,
                                IPSET_NONSTANDARDDATA,
                                IPPARM_H221NONSTANDARD,
                                (unsigned char)sizeof(IP_H221NONSTANDARD),
                                &appH221NonStd);
    }

    else
    {
        gc_util_insert_parm_ref(&pParmBlock,
                                IPSET_NONSTANDARDDATA,
                                IPPARM_NONSTANDARDDATA_OBJID,
                                (unsigned char)(strlen(pOid)+1),
                                pOid);
    }
```

4.3.4 Specifying Nonstandard Control Information When Using H.323

To specify Nonstandard Control information, use the **gc_SetUserInfo( )** function with a **duration** parameter set to GC_SINGLECALL to set nonstandard control information. If the **duration** parameter is set to GC_ALLCALLS, the function will fail.

Set up the GC_PARM_BLK pointed by the **infoparmblkp** function parameter with the IPSET_NONSTANDARDCONTROL parameter set ID and one of two combinations of parameter IDs. First set the IPPARM_NONSTANDARDDATA_DATA parameter ID (maximum length is MAX_NS_PARM_DATA_LENGTH or 128). Then set either of the following parameter IDs according to which type of object identifier to use:

- IPPARM_NONSTANDARDDATA_OBJID. The maximum length is MAX_NS_PARM_OBJID_LENGTH (40).
- IPPARM_H221NONSTANDARD

See Section 8.2.15, “IPSET_NONSTANDARDCONTROL”, on page 225 for more information.

The following code example shows how to set nonstandard data elements:

```
IP_H221NONSTANDARD appH221NonStd;
appH221NonStd.country_code = 181;
appH221NonStd.extension = 31;
appH221NonStd.manufacturer_code = 11;
char* pControl = "Control String";
char* pOid = "1 22 333 4444";
choiceOfNSControl = 1; /* App decides which type of object identifier to use */

/* setting NS Control */
gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_NONSTANDARDCONTROL,
                        IPPARM_NONSTANDARDDATA_DATA,
                        (unsingned char)(strlen(pControl)+1),
                        pControl);

    if (choiceOfNSControl)  /* App decide the CHOICE of OBJECTIDENTIFIER.
                               It cannot set both objid & h221 */
    {
        gc_util_insert_parm_ref(&pParmBlock,
                                IPSET_NONSTANDARDCONTROL,
                                IPPARM_H221NONSTANDARD,
                                (unsingned char)sizeof(IP_H221NONSTANDARD),
                                &appH221NonStd);
    }

    else
    {
        gc_util_insert_parm_ref(&pParmBlock,
                                IPSET_NONSTANDARDCONTROL,
                                IPPARM_NONSTANDARDDATA_OBJID,
                                (unsingned char)(strlen(pOid)+1),
                                pOid);
    }
```

4.3.5 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 endpoint.

Note: When using SIP, reasons are only supported when a call is disconnected while in the Offered state.

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

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

4.3.6 Setting Busy Reason Codes

Both SIP and H.323 define request response codes that can be included in the failure response messages that are sent when a local system cannot take additional incoming sessions. Global Call allows applications to set SIP and H.323 busy code values on a virtual board level.

SIP and H.323 busy codes are configured independently, and the configuration of each may be changed at any time. The busy codes are configured by calling **gc_SetConfigData( )** using the following parameter set ID and parameter ID:

- for SIP: IPSET_SIP_RESPONSE_CODE and IPPARM_BUSY_REASON; see Section 8.2.20, "IPSET_SIP_RESPONSE_CODE", on page 229.
- for H.323: IPSET_H323_RESPONSE_CODE and IPPARM_BUSY_CAUSE; see Section 8.2.8, "IPSET_H323_RESPONSE_CODE", on page 222.

4.3.6.1 Setting SIP Busy Code

For SIP, RFC3261 defines three applicable busy codes:

480 Temporarily Unavailable

The callee's end system was contacted successfully, but the callee is currently unavailable. For example, the callee may be not logged in, may be in a state that precludes communication, or may have activated the "do not disturb" feature. This busy code is also returned by a redirect or proxy server that recognizes the user identified by the Request-URI but does not currently have a valid forwarding location for that user.

486 Busy Here

The callee's end system was contacted successfully, but the callee is currently not willing or able to take additional calls at this end system. This response should be used if the user could be available elsewhere.

600 Busy Everywhere

The callee's end system was contacted successfully, but the callee is busy and does not wish to take the call at this time. This response should be used if the callee knows that no other end system will be available to accept this call.

By default, Global Call automatically responds with a 486 Busy Here when additional incoming call requests arrive after the maximum number of SIP calls per virtual board has been reached. A 480 Temporarily Unavailable or 600 Busy Everywhere reason code may be used instead of the 486 Busy Here if the application explicitly configures the busy code.

To configure the SIP busy reason code, call **gc_SetConfigData( )** using the parameter set ID IPSET_SIP_RESPONSE_CODE and the parameter ID IPPARM_BUSY_REASON as shown in the following code snippet:

```
#include "gclib.h"
   .
   .
/* configure SIP Busy Reason Code to 480 Temporarily Available */

GC_PARM_BLKP pParmBlock = NULL;

gc_util_set_insert_parm_val(&pParmBlock,
                            IPSET_SIP_RESPONSE_CODE,
                            IPPARM_BUSY_REASON,
                            sizeof(unsigned short),
                            IPEC_SIPReasonStatus480TemporarilyUnavailable);
```

```
gc_SetConfigData(GCTGT_CCLIB_NETIF,
                 board,
                 pParmBlock,
                 0,
                 GCUPDATE_IMMEDIATE,
                 &t,
                 EV_ASYNC);

gc_util_delete_parm_blk(pParmBlock);
```

4.3.6.2 Setting H.323 Busy Code

ITU Recommendation Q.850 defines cause codes that are used for H.323. Among the applicable busy cause definitions are:

Cause 34: No circuit/channel available
: Indicates there is no appropriate circuit/channel currently available to handle the call.

Cause 47: Resource unavailable/unspecified
: Indicates the resource is unavailable when no other cause values in the resource class applies.

To configure the H.323 busy reason code, call **gc_SetConfigData( )** using the parameter set ID IPSET_H323_RESPONSE_CODE and the parameter ID IPPARM_BUSY_CAUSE as shown in the following code snippet:

```
#include "gclib.h"
   .
   .
/* configure H.323 Busy Reason Code to 34 -  "No Circuit/Channel Available" */

GC_PARM_BLKP pParmBlock = NULL;

gc_util_set_insert_parm_val(&pParmBlock,
                            IPSET_H323_RESPONSE_CODE,
                            IPPARM_BUSY_CAUSE,
                            sizeof(unsigned short),
                            IPEC_Q931Cause34NoCircuitChannelAvailable);

gc_SetConfigData(GCTGT_CCLIB_NETIF,
                 board,
                 pParmBlock,
                 0,
                 GCUPDATE_IMMEDIATE,
                 &t,
                 EV_ASYNC);

gc_util_delete_parm_blk(pParmBlock);
```

4.4 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:

1. 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.

2. Use the **gc_Extension( )** function to request the data. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. 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).
3. A GCEV_EXTENSIONCMPLT event is generated in response to the **gc_Extension( )** request. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSIONCMPLT event is a pointer to an EXTENSIONEVTBLK structure that contains a GC_PARM_BLK with the requested call-related information.
4. Extract the information from the GC_PARM_BLK associated with the GCEV_EXTENSIONCMPLT event. In this case, the GC_PARM_BLK contains real data; that is, the value_size field is not 0, and includes the size of the data following for each parameter requested.

Table 4 shows the parameters that can be retrieved and when the information should be retrieved. The table also identifies which information can be retrieved when using H.323 and which information can be retrieved using SIP. For more information on individual parameters, refer to the corresponding parameter set reference section in Chapter 8, “IP-Specific Parameters”.

Table 4. Retrievable Call Information

Parameter	Set ID and Parameter ID(s)	When Information Can Be Retrieved	Datatype in value_buf Field (see Note 1)	SIP/ H.323
Call ID	IPSET_CALLINFO • IPPARM_CALLID	Any state after Offered or Proceeding	For SIP: string, max. length = MAX_IP_SIP_ CALLID_LENGTH For H.323: array of octets, length = MAX_IP_H323_ CALLID_LENGTH If protocol is unkown, MAX_IP_CALLID_ LENGTH defines the maximum Call ID length for any possible protocol.	both
Bearer Capability IE	IPSET_CALLINFO • IPPARM_BEARERCAP	After Offered	String, max. length = 255	H.323 only
Call Duration	IPSET_CALLINFO • IPPARM_CALL_DURATION	After Disconnected, before Idle	Unsigned long (value in ms)	H.323 only

Notes:
1. This field is the value_buf field in the GC_PARM_DATA structure associated with the GCEV_EXTENSIONCMPLT event generated in response to the **gc_Extension( )** function requesting the information.
2. Display information, user to user information, phone list, nonstandard data, vendor information and nonstandard control information, and H221 nonstandard information may not be present.
3.Vendor information is included in a Q931 SETUP message received from a peer.
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.

Table 4. Retrievable Call Information (Continued)

Parameter	Set ID and Parameter ID(s)	When Information Can Be Retrieved	Datatype in value_buf Field (see Note 1)	SIP/ H.323
Conference Goal	IPSET_CONFERENCE • IPPARM_CONFERENCE_GOAL	Any state after Offered or Proceeding	Uint[8]	H.323 only
Conference ID	IPSET_CONFERENCE • IPPARM_CONFERENCE_ID	Any state after Offered or Proceeding	char*, max. length = IP_CONFER ENCE_ID_ LENGTH (16)	H.323 only
Display Information	IPSET_CALLINFO • IPPARM_DISPLAY	Any state after Offered or Proceeding	char*, max. length = MAX_DISPLAY_ LENGTH (82), null-terminated	both
Facility IE	IPSET_CALLINFO • IPPARM_FACILITY	After Offered (SETUP message), Connected (CONNECT message), or the reception of a Facility message	String, max. length = 255	H.323 only
Nonstandard Control	IPSET_NONSTANDARDCONTROL • IPPARM_ NONSTANDARDDATA_DATA • IPPARM_ NONSTANDARDDATA_OBJID or • IPPARM_H221NONSTANDARD	See Section 4.4.1, “Retrieving Nonstandard Data From Protocol Messages When Using H.323”, on page 80 for more information.	char*, max. length = MAX_NS_PARM_ DATA_LENGTH (128) char*, max. length = MAX_NS_PARM_ OBJID_LENGTH (40)	H.323 only
Nonstandard Data	IPSET_NONSTANDARDDATA • IPPARM_ NONSTANDARDDATA_DATA • IPPARM_ NONSTANDARDDATA_OBJID or • IPPARM_H221NONSTANDARD	See Section 4.4.1, “Retrieving Nonstandard Data From Protocol Messages When Using H.323”, on page 80 for more information.	char*, max. length = MAX_NS_PARM_ DATA_LENGTH (128) char*, max. length = MAX_NS_PARM_ OBJID_LENGTH (40)	H.323 only
Phone List	IPSET_CALLINFO • IPPARM_PHONELIST	Any state after Offered or Proceeding	char*, max. length = 131	both
User to User Information	IPSET_CALLINFO • IPPARM_USERUSER_INFO	Any state after Offered or Proceeding	char*, max. length = MAX_USERUSER_IN FO_LENGTH (131 octets)	H.323 only

Notes:
1. This field is the value_buf field in the GC_PARM_DATA structure associated with the GCEV_EXTENSIONCMPLT event generated in response to the **gc_Extension( )** function requesting the information.
2. Display information, user to user information, phone list, nonstandard data, vendor information and nonstandard control information, and H221 nonstandard information may not be present.
3.Vendor information is included in a Q931 SETUP message received from a peer.
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.

Table 4. Retrievable Call Information (Continued)

Parameter	Set ID and Parameter ID(s)	When Information Can Be Retrieved	Datatype in value_buf Field (see Note 1)	SIP/ H.323
Vendor Product ID	IPSET_VENDORINFO • IPPARM_ VENDOR_PRODUCT_ID	Any state after Offered or Proceeding	char*, max. length = MAX_PRODUCT_ ID_LENGTH (32)	H.323 only
Vendor Version ID	IPSET_VENDORINFO • IPPARM_ VENDOR_VERSION_ID	Any state after Offered or Proceeding	char*, max. length = MAX_VERSION_ ID_LENGTH (32)	H.323 only
H.221 Nonstandard Information	IPSET_VENDORINFO • IPPARM_H221NONSTD	Any state after Offered or Proceeding	IP_H221_ NONSTANDARD (see note 4)	H.323 only

Notes:
1. This field is the value_buf field in the GC_PARM_DATA structure associated with the GCEV_EXTENSIONCMPLT event generated in response to the **gc_Extension( )** function requesting the information.
2. Display information, user to user information, phone list, nonstandard data, vendor information and nonstandard control information, and H221 nonstandard information may not be present.
3.Vendor information is included in a Q931 SETUP message received from a peer.
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 GCEV_EXTENSIONCMPLT 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.

4.4.1 Retrieving Nonstandard Data From Protocol Messages When Using H.323

Any Q.931 message can include nonstandard data. The application can use the **gc_Extension( )** function with and **ext_id** of IPEXTID_GETINFO to retrieve the data while a call is in any state. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. The information is included with the corresponding GCEV_EXTENSIONCMPLT termination event.

Note: When retrieving nonstandard data, it is only necessary to specify IPPARM_NONSTANDARDDATA_DATA in the extension request. It is not necessary to specify IPPARM_NONSTANDARDDATA_OBJID or IPPARM_H221NONSTANDARD. The call control library ensures that the GCEV_EXTENSIONCMPLT event includes the correct information.

4.4.2 Examples of Retrieving Call-Related Information

The following code demonstrates how to do the following:

- create a structure that identifies which information should be retrieved, then use the **gc_Extension( )** with an **extID** of IPEXTID_GETINFO to issue the request
- extract the data from a structure associated with the GCEV_EXTENSIONCMPLT event received as a termination event to the **gc_Extension( )** function

Similar code can be used when using SIP, except that the code must include only information parameters supported by SIP (see Table 4, "Retrievable Call Information", on page 78).

Specifying Call-Related Information to Retrieve

The following function shows how an application can construct and send a request to retrieve call-related information.

```
int getInfoAsync(CRN crn)
{
    GC_PARM_BLKP gcParmBlk = NULL;
    GC_PARM_BLKP retParmBlk;
    int frc;

    frc = gc_util_insert_parm_val(&gcParmBlk,
                                  IPSET_CALLINFO,
                                  IPPARM_PHONELIST,
                                  sizeof(int),1);
    if (GC_SUCCESS != frc)
    {
        return GC_ERROR;
    }

    frc = gc_util_insert_parm_val(&gcParmBlk,
                            IPSET_CALLINFO,
                            IPPARM_CALLID,
                            sizeof(int),1);
    if (GC_SUCCESS != frc)
    {
        return GC_ERROR;
    }

    frc = gc_util_insert_parm_val(&gcParmBlk,
                            IPSET_CONFERENCE,
                            IPPARM_CONFERENCE_ID,
                            sizeof(int),1);
    if (GC_SUCCESS != frc)
    {
        return GC_ERROR;
    }

    frc = gc_util_insert_parm_val(&gcParmBlk,
                            IPSET_CONFERENCE,
                            IPPARM_CONFERENCE_GOAL,
                            sizeof(int),1);
    if (GC_SUCCESS != frc)
    {
        return GC_ERROR;
    }
```

```
frc = gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_CALLINFO,
                        IPPARM_DISPLAY,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_CALLINFO,
                        IPPARM_USERUSER_INFO,
                        sizeof(int),1);

if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_VENDORINFO,
                        IPPARM_VENDOR_PRODUCT_ID,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_VENDORINFO,
                        IPPARM_VENDOR_VERSION_ID,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_VENDORINFO,
                        IPPARM_H221NONSTD,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,/* NS Data: setting this IPPARM implies
                                            retrieval of the complete element */
                        IPSET_NONSTANDARDDATA,
                        IPPARM_NONSTANDARDDATA_DATA,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}

frc = gc_util_insert_parm_val(&gcParmBlk,/* NS Control: setting this IPPARM implies
                                            retrieval of the complete element */
                        IPSET_NONSTANDARDCONTROL,
                        IPPARM_NONSTANDARDDATA_DATA,
                        sizeof(int),1);
if (GC_SUCCESS != frc)
{
    return GC_ERROR;
}
```

```
    frc = gc_Extension(GCTGT_GCLIB_CRN,
                       crn,
                       IPEXTID_GETINFO,
                       gcParmBlk,
                       &retParmBlk,
                       EV_ASYNC);
    if (GC_SUCCESS != frc)
    {
        return GC_ERROR;
    }

    gc_util_delete_parm_blk(gcParmBlk);
    return GC_SUCCESS;
}
```

Extracting Call-Related Information Associated with an Extension Event

The following code demonstrates how an application can extract call information when a GCEV_EXTENSIONCMPLT event is received as a result of a request for call-related information.

```
int OnExtensionAndComplete(GC_PARM_BLKP parm_blk,CRN crn)
{
   GC_PARM_DATA  *parmp = NULL;
   parmp = gc_util_next_parm(parm_blk,parmp);
   if (!parmp)
   {
      return GC_ERROR;
   }

   while (NULL != parmp)
   {
      switch (parmp->set_ID)
      {
         case IPSET_CALLINFO:
            switch (parmp->parm_ID)
            {
                case IPPARM_DISPLAY:
                   if(parmp->value_size != 0)
                   {
                      printf("\tReceived extension data DISPLAY: %s\n", parmp->value_buf);
                   }
                   break;

                case IPPARM_CALLID:
                   /* print the Call ID in parmp->value_buf as array of bytes */
                   for (int count = 0; count < parmp->value_size; count++)
                   {
                      printf("0x%2X ", value_buf[count]);
                   }
                   break;

                case IPPARM_USERUSER_INFO:
                   if(parmp->value_size != 0)
                   {
                      printf("\tReceived extension data UUI: %s\n", parmp->value_buf);
                   }
                   break;
```

```
         case IPPARM_PHONELIST:
            if(parmp->value_size != 0)
            {
               printf("\tReceived extension data PHONELIST: %s\n",
                        parmp->value_buf);
            }
            break;

         default:
            printf("\tReceived unknown CALLINFO extension parmID %d\n",
                     parmp->parm_ID);
            break;
      }/* end switch (parmp->parm_ID) for IPSET_CALLINFO */
      break;

   case IPSET_CONFERENCE:
      switch (parmp->parm_ID)
      {
         case IPPARM_CONFERENCE_GOAL:
            if(parmp->value_size != 0)
            {
               printf("\tReceived extension data IPPARM_CONFERENCE_GOAL: %d\n",
                        (unsigned int)(*(parmp->value_buf)));
            }
            break;

         case IPPARM_CONFERENCE_ID:
            if(parmp->value_size != 0)
            {
               printf("\tReceived extension data IPPARM_CONFERENCE_ID: %s\n",
                        parmp->value_buf);
            }
            break;

         default:
            printf("\tReceived unknown CONFERENCE extension parmID %d\n",
                     parmp->parm_ID);
            break;
      }
      break;

   case IPSET_VENDORINFO:
      switch (parmp->parm_ID)
      {
         case IPPARM_VENDOR_PRODUCT_ID:
            if(parmp->value_size != 0)
            {
               printf("\tReceived extension data  PRODUCT_ID %s\n", parmp->value_buf);
            }
            break;

         case IPPARM_VENDOR_VERSION_ID:
            if(parmp->value_size != 0)
            {
               printf("\tReceived extension data  VERSION_ID %s\n", parmp->value_buf);
            }
            break;

         case IPPARM_H221NONSTD:
         {
            if(parmp->value_size == sizeof(IP_H221NONSTANDARD))
            {
               IP_H221NONSTANDARD *pH221NonStandard;
               pH221NonStandard = (IP_H221NONSTANDARD *)(&(parmp->value_buf));
               printf("\tReceived extension data VENDOR H221NONSTD:
                        CC=%d, Ext=%d, MC=%d\n",
                        pH221NonStandard->country_code,
```

```
                    pH221NonStandard->extension,
                    pH221NonStandard->manufacturer_code);
          }
      }
          break;

      default:
         printf("\tReceived unknown VENDORINFO extension parmID %d\n",
                  parmp->parm_ID);
         break;
   }/* end switch (parmp->parm_ID) for IPSET_VENDORINFO */
   break;

case IPSET_NONSTANDARDDATA:
   switch (parmp->parm_ID)
   {
      case IPPARM_NONSTANDARDDATA_DATA:
         printf("\tReceived extension data (NSDATA) DATA: %s\n", parmp->value_buf);
         break;

      case IPPARM_NONSTANDARDDATA_OBJID:
         printf("\tReceived extension data (NSDATA) OBJID: %s\n", parmp->value_buf);
         break;

      case IPPARM_H221NONSTANDARD:
      {
         if(parmp->value_size == sizeof(IP_H221NONSTANDARD))
         {
            IP_H221NONSTANDARD *pH221NonStandard;
            pH221NonStandard = (IP_H221NONSTANDARD *)(&(parmp->value_buf));
            printf("\tReceived extension data (NSDATA) h221:CC=%d, Ext=%d, MC=%d\n",
                    pH221NonStandard->country_code,
                    pH221NonStandard->extension,
                    pH221NonStandard->manufacturer_code);
         }
      }
      break;

      default:
         printf("\tReceived unknown (NSDATA) extension parmID %d\n",
                 parmp->parm_ID);
         break;
   }
   break;

case IPSET_NONSTANDARDCONTROL:
   switch (parmp->parm_ID)
   {
      case IPPARM_NONSTANDARDDATA_DATA:
         printf("\tReceived extension data (NSCONTROL) DATA: %s\n",
                 parmp->value_buf);
         break;

      case IPPARM_NONSTANDARDDATA_OBJID:
         printf("\tReceived extension data (NSCONTROL) OBJID: %s\n",
                 parmp->value_buf);
         break;

      case IPPARM_H221NONSTANDARD:
      {
         if(parmp->value_size == sizeof(IP_H221NONSTANDARD))
         {
            IP_H221NONSTANDARD *pH221NonStandard;
            pH221NonStandard = (IP_H221NONSTANDARD *)(&(parmp->value_buf));
            printf("\tReceived extension data (NSCONTROL) h221:CC=%d, Ext=%d, MC=%d\n",
                    pH221NonStandard->country_code,
```

```
                        pH221NonStandard->extension,
                        pH221NonStandard->manufacturer_code);
            }
         }
         break;

         default:
            printf("\tReceived unknown (NSCONTROL) extension parmID %d\n",
                    parmp->parm_ID);
            break;
      }
      break;

   case IPSET_MSG_Q931:
      switch (parmp->parm_ID)
      {
         case IPPARM_MSGTYPE:
            switch ((*(int *)(parmp->value_buf)))
            {
               case IP_MSGTYPE_Q931_FACILITY:
                  printf("\tReceived extension data IP_MSGTYPE_Q931_FACILITY\n");
                  break;

               default:
                  printf("\tReceived unknown MSG_Q931 extension parmID %d\n",
                          parmp->parm_ID);
                  break;
            } /* end  switch ((int)(parmp->value_buf)) */
              break;
      }/* end switch (parmp->parm_ID) for IPSET_MSG_Q931 */
      break;

   case IPSET_MSG_H245:
      switch (parmp->parm_ID)
      {
         case IPPARM_MSGTYPE:
            switch ((*(int *)(parmp->value_buf)))
            {
               case IP_MSGTYPE_H245_INDICATION:
                  printf("\tReceived extension data IP_MSGTYPE_H245_INDICATION\n");
                  break;

               default:
                  printf("\tReceived unknown MSG_H245 extension parmID %d\n",
                          parmp->parm_ID);
                  break;
            }/* end  switch ((int)(parmp->value_buf)) */
             break;
      }/* end switch (parmp->parm_ID) for IPSET_MSG_H245 */
      break;

   default:
      printf("\t Received unknown extension setID %d\n",parmp->set_ID);
      break;
}/* end switch (parmp->set_ID) */

parmp = gc_util_next_parm(parm_blk,parmp);
}

return GC_SUCCESS;
}
```

Note: IPPARM_CALLID is a set of bytes and should *not* be interpreted as a string.

Retrieving Call ID

The following code example illustrates how to request Call ID information via a **gc_Extension( )** call.

```
/*
 * Assume the following has been done:
 * 1. device has been opened (e.g. :N_iptB1T1:P_SIP, :N_iptB1T2:P_SIP, etc...)
 * 2. gc_WaitCall() has been issued to wait for a call.
 * 3. gc_GetMetaEvent() or gc_GetMetaEventEx() (Windows) has been called
 *    to convert the event into metaevent.
 * 4. a GCEV_OFFERED has been detected.
 */

#include <stdio.h>
#include <srllib.h>
#include <gclib.h>
#include <gcerr.h>
#include <gcip.h>

/*
 * Assume the 'crn' parameter holds the CRN associated
 * with the detected GCEV_OFFERED event.
 */
int request_call_info(CRN crn)
{
    int retval = GC_SUCCESS;
    GC_PARM_BLKP parmblkp = NULL;  /* input parameter block pointer */
    GC_PARM_BLKP retblkp = NULL;   /* pointer for output parameter block (unused) */
    GC_INFO gc_error_info;         /* GlobalCall error information data */

    /* allocate GC_PARM_BLK for Call-ID message parameter */
    gc_util_insert_parm_val(&parmblkp, IPSET_CALLINFO, IPPARM_CALLID, sizeof(int), 1);
    if (parmblkp == NULL)
    {
        /* memory allocation error */
        return(-1);
    }

    /* retrieve the Call-ID from the network */
    if (gc_Extension(GCTGT_GCLIB_CRN, crn, IPEXTID_GETINFO, parmblkp, &retblkp,
                     EV_ASYNC) != GC_SUCCESS)
    {
        /* process error return as shown */
        gc_ErrorInfo( &gc_error_info );
        printf ("Error: gc_Extension() on crn: 0x%lx, GC ErrorValue: 0x%hx - %s,
                CCLibID: %i - %s, CC ErrorValue: 0x%lx - %s\n",
                crn, gc_error_info.gcValue, gc_error_info.gcMsg, gc_error_info.ccLibId,
                gc_error_info.ccLibName, gc_error_info.ccValue, gc_error_info.ccMsg);
    }

    /* free the parameter block */
    gc_util_delete_parm_blk(parmblkp);

    return (retval);
}
```

Parsing Call ID Information (SIP Protocol)

The following code example illustrates how to parse the Call ID information retrieved via a **gc_Extension( )** call when the SIP protocol is being used.

```
/*
 * Assume the following has been done:
 * 1. device has been opened (e.g. :N_iptB1T1:P_SIP, :N_iptB1T2:P_SIP, etc...)
 * 2. gc_GetMetaEvent() or gc_GetMetaEventEx() (Windows) has been called
 *    to convert the event into metaevent.
 * 3. a GCEV_EXTENSIONCMPLT has been detected.
 */

#include <stdio.h>
#include <srllib.h>
#include <gclib.h>
#include <gcerr.h>
#include <gcip.h>

/*
 * Assume the 'crn' parameter holds the CRN associated with the detected
 * GCEV_EXTENSIONCMPLT event, and the 'pEvt' parameter holds a pointer
 * to the detected metaevent.
 */

int print_call_info(CRN crn, METAEVENT *pEvt)
{
    EXTENSIONEVTBLK *ext_data = NULL;
    GC_PARM_DATA *parmp = NULL;
    GC_PARM_BLK  *parm_blkp;

    if (pEvt)
    {
        if (pEvt->evttype == GCEV_EXTENSIONCMPLT)
        {
            ext_data = (EXTENSIONEVTBLK *)(pEvt->extevtdatap);
        }
    }

    if (!ext_data)
    {
        printf("\tNot a GCEV_EXTENSIONCMPLT event.\n");
        return GC_ERROR;
    }

    parm_blk = &(ext_data->parmblk);

    parmp = gc_util_next_parm(parm_blkp,parmp);
    if (!parmp)
    {
        printf("\tNo data returned in extension event for crn: 0x%lx\n", crn);
        return GC_ERROR;
    }

    while (NULL != parmp)
    {
        switch (parmp->set_ID)
        {
            case IPSET_CALLINFO:
                switch (parmp->parm_ID)
                {
                    case IPPARM_CALLID:
                        if(parmp->value_size != 0)
                        {
                            /* Here's where we print the SIP Call ID */
                            printf("\tReceived extension data IPPARM_CALLID: %s\n",
                                   parmp->value_buf);
                        }
                    break;
```

```
                    default:
                        printf("\tReceived unexpected IPSET_CALLINFO parmID %d\n",
                               parmp->parm_ID);
                    break;
                } /* end switch (parmp->parm_ID) */
                break;

            default:
                printf("\t Received unexpected extension setID %d\n",
                       parmp->set_ID);
            break;

        } /* end switch (parmp->set_ID) */

        parmp = gc_util_next_parm(parm_blkp,parmp);
    } /* end while (parmp != NULL) */

    return GC_SUCCESS;
}
```

4.5 Setting and Retrieving Q.931 Message IEs

Global Call supports the setting and retrieving of Information Elements (IEs) in selected Q.931 messages. The level of support is described in the following topics:

- Enabling Access to Q.931 Message IEs
- Supported Q.931 Message IEs
- Setting Q.931 Message IEs
- Retrieving Q.931 Message IEs
- Common Usage Scenarios Involving Q.931 Message IEs

4.5.1 Enabling Access to Q.931 Message IEs

The ability to set and retrieve Q.931 message IEs is an optional feature that can be enabled or disabled at the time the **gc_Start( )** function is called.

The **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** functions, which must be called before **gc_Start( )**, populate the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default values. The default value of the h323_msginfo_mask field in the IP_VIRTBOARD structure disables access to Q.931 message information elements. The default value of the h323_msginfo_mask field must therefore be overridden with the value IP_H323_MSGINFO_ENABLE for each IPT board device on which the feature is to be enabled. The following code snippet provides an example:

```
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE; /* override Q.931 message default */
ip_virtboard[1].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE; /* override Q.931 message default */
```

Note: Setting the h323_msginfo_mask field to a value of IP_H323_MSGINFO_ENABLE enables the setting or retrieving of all supported Q.931 message information elements collectively. Enabling and disabling access to individual Q.931 message information elements is **not** supported.

4.5.2 Supported Q.931 Message IEs

Table 5 shows the supported Q.931 message Information Elements (IEs), the parameter set ID and parameter ID that should be included in a GC_PARM_BLK when setting or retrieving the IEs, and the maximum allowed length of the IE value.

Table 5. Supported Q.931 Message Information Elements

IE Name	Set/Get	Set ID	Parameter ID	Maximum Length
Bearer Capability	Get and Set	IPSET_CALLINFO	IPPARM_BEARERCAP	255
Facility	Get and Set	IPSET_CALLINFO	IPPARM_FACILITY	255
Note: These parameters are character arrays with the maximum size of the array equal to the maximum length shown.				

4.5.3 Setting Q.931 Message IEs

The Global Call library supports the setting of the following Information Elements (IEs) in the following *outgoing* Q.931 messages:

- Bearer Capability IE in a SETUP message
- Facility IE in SETUP, CONNECT, and FACILITY messages

The **gc_SetUserInfo( )** function is used to set these IEs. The appropriate function parameters in this context are:

- **target_type** – GCTGT_GCLIB_CHAN
- **target_id** – line device
- **infoparmblkp** – a GC_PARM_BLK containing the IPSET_CALLINFO parameter set ID and one of the following parameter IDs:
  - IPPARM_BEARERCAP
  - IPPARM_FACILITY
- **duration** – GC_SINGLECALL (GC_ALLCALLS is not supported in this context)

4.5.4 Retrieving Q.931 Message IEs

The Global Call library supports the retrieval of the following Information Elements (IEs) from the following *incoming* Q.931 messages:

- Bearer Capability IE in a SETUP message
- Facility IE in SETUP, CONNECT, and FACILITY messages

Table 6 shows the Global Call events generated for incoming Q.931 messages and the parameter set ID and parameter IDs contained in the GC_PARM_BLK associated with each event.

Table 6. Supported IEs in Incoming Q.931 Messages

Incoming Q.931 Message	Global Call Event	Set ID	Parm ID
SETUP	GCEV_OFFERED	IPSET_CALLINFO	IPPARM_BEARERCAP
SETUP	GCEV_OFFERED	IPSET_CALLINFO	IPPARM_FACILITY
CONNECT	GCEV_CONNECTED	IPSET_CALLINFO	IPPARM_FACILITY
FACILITY	GCEV_EXTENSION with an ext_id of EXTID_RECEIVEMSG	IPSET_CALLINFO	IPPARM_FACILITY

Note: The application must retrieve the necessary IEs by copying them into its own buffer before the next call to **gc_GetMetaEvent( )**. Once the next **gc_GetMetaEvent( )** call is issued, the SIP information will no longer be available.

4.5.5 Common Usage Scenarios Involving Q.931 Message IEs

Table 7 shows how Global Call handles common scenarios that involve the use of Q.931 message IEs.

Table 7. Common Usage Scenarios Involving Q.931 Message IEs

Scenario	Behavior
The application invokes **gc_SetUserInfo( )** to set the Bearer Capability IE, then invokes **gc_MakeCall( )**	The Bearer Capability IE is parsed and added to the new outgoing SETUP message.
The application invokes **gc_SetUserInfo( )** to set the Facility IE, then invokes **gc_MakeCall( )**	The Facility IE is parsed and added to the new outgoing SETUP message.
The application invokes **gc_SetUserInfo( )** to set the Bearer Capability IE and the Facility IE, then invokes **gc_MakeCall( )**	The Bearer Capability IE and the Facility IE are parsed and added to the new outgoing SETUP message.
The application invokes **gc_SetUserInfo( )** to set the Facility IE, then invokes **gc_AnswerCall( )**	The Facility IE is parsed and added to the new outgoing CONNECT message.
The application invokes **gc_SetUserInfo( )** to set the Facility IE, then invokes **gc_Extension( )**	The Facility IE is parsed and added to the new outgoing FACILITY message.
The application receives a GCEV_OFFERED event with a Bearer Capability IE	The application retrieves the Bearer Capability IE using **gc_GetMetaEvent( )** and **gc_util_next_parm( )**.
The application receives a GCEV_OFFERED event with a Facility IE	The application retrieves the Facility IE using **gc_GetMetaEvent( )** and **gc_util_next_parm( )**.
The application receives a GCEV_OFFERED event with Bearer Capability IE and Facility IE	The application retrieves the Bearer Capability IE and Facility IE using **gc_GetMetaEvent( )** and **gc_util_next_parm( )**.
The application receives a GCEV_CONNECTED event with a Facility IE	The application retrieves the Facility IE using **gc_GetMetaEvent( )** and **gc_util_next_parm( )**.
The application receives a GCEV_EXTENSION event with a Facility IE	The application retrieves the Facility IE using **gc_GetMetaEvent( )** and **gc_util_next_parm( )**.

4.6 Setting and Retrieving SIP Message Information Fields

Global Call supports the setting and retrieving of SIP message information fields in the INVITE message. This feature is described in the following topics:

- Enabling Access to SIP Message Information Fields
- Supported SIP Message Information Fields
- Setting a SIP Message Information Field
- Retrieving a SIP Message Information Field

4.6.1 Enabling Access to SIP Message Information Fields

The ability to set and retrieve SIP message information fields is an optional feature that can be enabled or disabled at the time the **gc_Start( )** function is called.

The **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** utility functions, which must be called before the **gc_Start( )** function, populate the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default values. The default value of the sip_msginfo_mask field in the IP_VIRTBOARD structure disables access to SIP message information fields. The default value of the sip_msginfo_mask field must be overridden with the value IP_SIP_MSGINFO_ENABLEfor each IPT board device on which the feature is to be enabled. The following code snippet provides an example:

```
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].sip_msginfo_mask = IP_SIP_MSGINFO_ENABLE; /* override SIP message default */
ip_virtboard[1].sip_msginfo_mask = IP_SIP_MSGINFO_ENABLE; /* override SIP message default */
```

Note: Setting the sip_msginfo_mask field to a value of IP_SIP_MSGINFO_ENABLE enables setting or retrieving all SIP message information fields collectively. Enabling and disabling access to individual SIP message information fields is **not** supported.

4.6.2 Supported SIP Message Information Fields

Table 8 shows the supported SIP message information fields for INVITE messages. The fields are set in a GC_PARM_BLK structure associated with the **gc_SetUserInfo( )** function and retrieved from a GC_PARM_BLK structure associated with a GCEV_OFFERED event. Table 8 also indicates the relevant parameter set ID and parameter ID for each supported message field and the defines that identify the maximum allowable length for each field.

Table 8. Supported SIP Message Information Fields

Field Name	Set/Get	Set ID & Parameter ID	Maximum Length Define †
Call ID ††	Set and Get	IPSET_SIP_MSGINFO • IPPARM_CALLINFO_HDR	IP_CALLID_HDR_MAXLEN
Contact Display String	Set and Get	IPSET_SIP_MSGINFO • IPPARM_CONTACT_DISPLAY	IP_CONTACT_DISPLAY_ MAXLEN
Contact URI	Set and Get	IPSET_SIP_MSGINFO • IPPARM_CONTACT_URI	IP_CONTACT_URI_MAXLEN
Diversion URI	Set and Get	IPSET_SIP_MSGINFO • IPPARM_DIVERSION_URI	IP_DIVERSION_MAXLEN
From Display String	Set and Get	IPSET_SIP_MSGINFO • IPPARM_FROM_DISPLAY	IP_FROM_DISPLAY_MAXLEN
Referred-by	Set and Get	IPSET_SIP_MSGINFO • IPPARM_REFERRED_BY	IP_REFERRED_BY_MAXLEN
Replaces	Set and Get	IPSET_SIP_MSGINFO • IPPARM_REPLACES	IP_REPLACES_MAXLEN
Request URI	Set and Get	IPSET_SIP_MSGINFO • IPPARM_REQUEST_URI	IP_REQURI_MAXLEN
To Display String	Set and Get	IPSET_SIP_MSGINFO • IPPARM_TO_DISPLAY	IP_TO_DISPLAY_MAXLEN

† These parameters are character arrays with the maximum size of the array (including the NULL) equal to the corresponding maximum length define.
†† Directly setting the Call ID using this parameter overrides any Call ID value that is set using the IPSET_CALLINFO / IPPARM_CALLID parameter.

Note: The From URI and To URI message fields are not discussed as part of this feature because they are accessible using other Global Call functions. For example, see Section 7.2.9, "gc_GetCallInfo( ) Variances for IP", on page 166, for information on retrieving information from the From URI and To URI message fields.

4.6.3 Setting a SIP Message Information Field

Use the **gc_SetUserInfo( )** function to set the value of a SIP message information field. The **duration** parameter must be set to GC_SINGLECALL when calling **gc_SetUserInfo( )** to set SIP message information fields. The information is not transmitted until the **gc_MakeCall( )** function is issued.

Note: Using the **gc_SetUserInfo( )** function to set SIP message information requires a detailed knowledge of the SIP protocol and its relationship to Global Call. The application has the responsibility to ensure that the correct SIP message information is set before calling the appropriate Global Call function.

Calling the **gc_SetUserInfo( )** function results in the following behavior:

- SIP message information fields that are set do not take effect until the **gc_MakeCall( )** function is issued.

- Using the **gc_SetUserInfo( )** does not affect incoming SIP messages on the same channel.
- Any SIP message information fields that are set only affect the next Global Call function call.
- The **gc_SetUserInfo( )** function fails with GC_ERROR if the sip_msginfo_mask field in the IP_VIRTBOARD structure is not set to IP_SIP_MSGINFO_ENABLE. When **gc_ErrorInfo( )** is called, the error code is IPERR_BAD_PARAM.

The following code shows how to set the Request URI information field before issuing **gc_MakeCall( )**. This translates to a SIP INVITE message with the specified request-URI.

```
#include "gclib.h"
..
..
GC_PARM_BLK  *pParmBlock = NULL;
char         *pDestAddrBlk = "1111@127.0.0.1\0";
char         *pReqURI = "sip:2222@127.0.0.1\0";


/* Insert SIP request URI field */
/* Add 1 to strlen for the NULL termination character */
gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_SIP_MSGINFO,
                        IPPARM_REQUEST_URI,
                        (unsigned char)strlen(pReqURI) + 1,
                        pReqURI);

/* Set Call Information */
gc_SetUserInfo(GCTGT_GCLIB_CHAN, ldev, pParmBlock, GC_SINGLECALL);

gc_util_delete_parm_blk(pParmBlock);

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_TRANSPARENT;

/* calling the function with the MAKECALL_BLK,
the INVITE "To" field will be: 1111@127.0.0.1
the INVITE RequestURI will be: sip:2222@127.0.0.1
*/
gc_MakeCall(ldev, &crn, NULL, &gcmkbl, MakeCallTimeout, EV_ASYNC);
```

The information fields that can be set are described in Table 8, "Supported SIP Message Information Fields", on page 93.

4.6.4 Retrieving a SIP Message Information Field

SIP information is reported to the application via standard Global Call events that are processed using the **gc_GetMetaEvent( )** function.

Note: The application must retrieve the necessary SIP message information by copying it into its own buffer before the next call to **gc_GetMetaEvent( )**. Once the next **gc_GetMetaEvent( )** call is issued, the SIP information will no longer be available.

The following code demonstrates how to copy the Request URI information field from the associated GCEV_OFFERED event. The GC_PARM_BLK structure containing the information field is referenced via the extevtdatap pointer in the METAEVENT structure. In this particular scenario, the GCEV_OFFERED event is generated as a result of receiving an INVITE message.

```
#include "gclib.h"
..
..
METAEVENT  metaevt;
GC_PARM_BLK   *pParmBlock = NULL;
GC_PARM_DATA  *parmp = NULL;
char          reqestURI[IP_REQUEST_URI_MAXLEN];

/* Get Meta Event */
gc_GetMetaEvent(&metaevt);

switch(metaevt->evttype)
   {
    .
    .
    .
   case GCEV_OFFERED:
      currentCRN = metaevt->crn;
      pParmBlock = (GC_PARM_BLK*)(metaevt->extevtdatap);
      parmp = NULL;

      /* going thru each parameter block data*/
      while ((parmp = gc_util_next_parm(pParmBlock,parmp)) != 0)
      {
         switch (parmp->set_ID)
         {
         /* Handle SIP message information */
            case IPSET_SIP_MSGINFO:
               switch (parmp->parm_ID)
               {
               /* Copy Request URI from parameter block */
               /* NOTE: value_size = string length + 1 (for the NULL termination) */
                  case IPPARM_REQUEST_URI:
                     strncpy(requestURI, parmp->value_buf, parmp->value_size);
                     break;
               }
         }
      break;
      }
    .
    .
    .
   }
```

The information fields that can be retrieved are described in Table 8, “Supported SIP Message Information Fields”, on page 93.

4.7 Handling DTMF

DTMF handling is described under the following topics:

- Specifying DTMF Support
- Getting Notification of DTMF Detection
- Generating DTMF
- Generating or Detecting DTMF Tones Using a Voice Resource

4.7.1 Specifying DTMF Support

Global Call can be used to configure which DTMF modes (UII Alphanumeric, RFC 2833, or Inband) are supported by the application. The DTMF mode can be specified in one of three ways:

- for all line devices simultaneously by using **gc_SetConfigData( )**
- on a per-line device basis by using **gc_SetUserInfo( )** with a **duration** parameter value of GC_ALLCALLS
- on a per-call basis by using **gc_SetUserInfo( )** with a **duration** parameter value of GC_SINGLECALL

The GC_PARM_BLK associated with the **gc_SetConfigData( )** or **gc_SetUserInfo( )** function should be used to discover which DTMF modes are supported. The GC_PARM_BLK should include the IPSET_DTMF parameter set ID and the IPPARM_SUPPORT_DTMF_BITMASK parameter ID, which specifies the DTMF transmission mode(s), with one of the following values:

IP_DTMF_TYPE_ALPHANUMERIC (default)
: For H.323, DTMF digits are sent and received in H.245 User Input Indication (UII) Alphanumeric messages.
For SIP, this value is **not** supported, and one of the following two options must therefore be explicitly specified.

IP_DTMF_TYPE_INBAND_RTP
: DTMF digits are sent and received inband via standard RTP transcoding.

IP_DTMF_TYPE_RFC_2833
: DTMF digits are send and received in the RTP stream as defined in RFC 2833.

SIP applications must change the default signaling mode to either in-band or RFC2833 prior to calling **gc_MakeCall( )**, **gc_AnswerCall( )**, **gc_AcceptCall( )**, and **gc_CallAck( )**. If a SIP application does not make this change, the functions will fail with an IPERR_NO_DTMF_CAPABILITY.

The following code snippet shows how to specify the out-of-band signaling mode:

```
{
    GC_PARM_BLKP parmblkp = NULL;
    gc_util_insert_parm_val(&parmblkp,
                            IPSET_DTMF,
                            IPPARM_SUPPORT_DTMF_BITMASK,
                            sizeof(char),
                            IP_DTMF_TYPE_INBAND_RTP);

    if (gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[callindex].ldev,
                       parmblkp, GC_ALLCALLS) != GC_SUCCESS) {

            // gc_SetUserInfo returned an error
}
gc_util_delete_parm_blk(parmblkp);
```

Note: The IPPARM_SUPPORT_DTMF_BITMASK can only be replaced; it cannot be modified. For each **gc_SetConfigData( )** or **gc_SetUserInfo( )** call, the IPPARM_SUPPORT_DTMF_BITMASK parameter is overwritten.

The mode in which DTMF is transmitted (Tx) is determined by the intersection of the mode values specified by the IPPARM_SUPPORT_DTMF_BITMASK and the receive capabilities of the remote endpoint. When this intersection includes multiple modes, the selected mode is based on the following priority:

1. RFC 2833
2. H.245 UII Alphanumeric (H.323 only)
3. Inband

The mode in which DTMF is received (Rx) is based on the selection of transmission mode from the remote endpoint; however, RFC 2833 can only be received if RFC 2833 is specified by the IPPARM_SUPPORT_DTMF_BITMASK parameter ID.

Table 9 summarizes the DTMF mode settings and associated behavior.

Table 9. Summary of DTMF Mode Settings and Behavior

IP_DTMF_TYPE_ RFC_2833	IP_DTMF_TYPE_ ALPHANUMERIC†	IP_DTMF_TYPE_ INBAND	Transmit (Tx) DTMF Mode	Receive (Rx) DTMF Mode
1 (enabled)	0 (disabled)	0 (disabled)	RFC 2833 if supported by remote endpoint, otherwise UII Alphanumeric†	RFC 2833, UII Alphanumeric† or Inband as chosen by the remote endpoint
0 (disabled)	1 (enabled)	0 (disabled)	UII Alphanumeric†	UII Alphanumeric† or Inband as chosen by the remote endpoint
0 (disabled)	0 (disabled)	1 (enabled)	Inband	UII Alphanumeric† or Inband as chosen by the remote endpoint
0 (disabled)	1 (enabled)	1 (enabled)	UII Alphanumeric†	UII Alphanumeric† or Inband as chosen by the remote endpoint
1 (enabled)	1 (enabled)	0 (disabled)	RFC 2833 if supported by remote endpoint, otherwise UII Alphanumeric†	RFC 2833, UII Alphanumeric† or Inband as chosen by the remote endpoint

† Applies to H.323 only.

Table 9. Summary of DTMF Mode Settings and Behavior (Continued)

IP_DTMF_TYPE_ RFC_2833	IP_DTMF_TYPE_ ALPHANUMERIC†	IP_DTMF_TYPE_ INBAND	Transmit (Tx) DTMF Mode	Receive (Rx) DTMF Mode
1 (enabled)	0 (disabled)	0 (disabled)	RFC 2833 if supported by remote endpoint, otherwise UII Alphanumeric†	RFC 2833, UII Alphanumeric† or Inband as chosen by the remote endpoint
1 (enabled)	0 (disabled)	1 (enabled)	RFC 2833 if supported by the remote endpoint, otherwise Inband	RFC 2833, UII Alphanumeric† or Inband as chosen by the remote endpoint
1 (enabled)	1 (enabled)	1 (enabled)	RFC 2833 if supported by the remote endpoint, otherwise UII Alphanumeric†	RFC 2833, UII Alphanumeric† or Inband as chosen by the remote endpoint
† Applies to H.323 only.				

When using RFC 2833, the payload type is specified using the IPSET_DTMF parameter set ID and the IPPARM_DTMF_RFC2833_PAYLOAD_TYP parameter ID with one of the following values:

- IP_USE_STANDARD_PAYLOADTYPE (default payload type, 101)
- Any value in the range 96 to 127 (dynamic payload type)

4.7.2 Getting Notification of DTMF Detection

Once DTMF support has been configured (see Section 4.7.1, “Specifying DTMF Support”, on page 96), the application can specify which DTMF modes will provide notification when DTMF digits are detected. The events for this notification must be enabled; see Section 4.11, “Enabling and Disabling Unsolicited Notification Events”, on page 105.

Once the events are enabled, when an incoming DTMF digit is detected, the application receives a GCEV_EXTENSION event, with an extID of IPEXTID_RECEIVE_DTMF. The GCEV_EXTENSION event contains the digit and the method. The GC_PARM_BLK associated with the event contains the IPSET_DTMF parameter set ID and the following parameter ID:

IPPARM_DTMF_ALPHANUMERIC
: For H.323, DTMF digits are received in H.245 User Input Indication (UII) alphanumeric messages. The parameter value is of type IP_DTMF_DIGITS. See the reference page for IP_DTMF_DIGITS on page 240 for more information. For SIP, this parameter is **not** supported.

4.7.3 Generating DTMF

Once DTMF support has been configured (see Section 4.7.1, "Specifying DTMF Support", on page 96), the application can use the **gc_Extension( )** function to generate DTMF digits. The relevant **gc_Extension( )** function parameter values in this context are:

- **target_type** should be GCTGT_GCLIB_CRN
- **target_id** should be the actual CRN
- **ext_ID** should be IPEXTID_SEND_DTMF

The GC_PARM_BLK pointed to by the **parmblkp** parameter must contain the IPSET_DTMF parameter set ID and the following parameter ID:

IPPARM_DTMF_ALPHANUMERIC
: For H.323, specifies that DTMF digits are to be sent in H.245 User Input Indication (UII) Alphanumeric messages. For SIP, this parameter is **not** supported.

4.7.4 Generating or Detecting DTMF Tones Using a Voice Resource

Using a voice resource to generate or detect DTMF tones in Inband or RFC2833 DTMF transfer mode requires that the voice resource (for example, dxxxB1C1) be attached to the IPT network device (for example, iptB1T1) that also has an IP Media device (ipmB1C1) attached. This can be achieved using the **gc_OpenEx( )** function as follows:

```
gc_OpenEx(lindevice, ":P_IP:N_iptB1T1:M_ipmB1C1:V_dxxxB1C1", EV_ASYNC, userattr)
```

where:

- linedevice - is a Global Call device
- P_IP - indicates that the device supports both the H.323 and SIP protocols
- N_iptB1T1 - identifies the IPT network device
- M_ipmB1C1 - identifies the IPT Media device
- V_dxxB1C1 - specifies the voice resource that will be used to generate or detect the DTMF tones
- EV_ASYNC - indicates the function operates in asynchronous mode
- userattr - points to a buffer where user information can be stored

Note: Alternatively, the IPT network device and IP Media device can be opened without the voice resource, and the IP line device can be routed to the voice device when needed.

Once the voice resource is attached to the IPT network and IPT Media devices, the following voice library functions can be used:

- **dx_dial( )** to generate DTMF tones
- **dx_getdig( )** to detect DTMF tones

4.8 Getting Media Streaming Status and Connection Information

The application can receive notification of changes in the status (connection and disconnection) of media streaming in the transmit and receive directions as GC_EXTENSIONEVT events. When the event is a notification of the connection of the media stream in either direction, information about the coders negotiated for that direction is also available.

The events for this notification must be enabled by setting or adding the bitmask value EXTENSIONEVT_SIGNALING_STATUS to the GC_EXTENSIONEVT mask; see Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105. Once the events are enabled, when a media streaming connection state changes, the application receives a GCEV_EXTENSION event. The EXTENSIONEVTBLK structure pointed to by the extevtdatap pointer within the GCEV_EXTENSION event will contain the following information for all media streaming status changes:

extID
: IPEXTID_MEDIAINFO

parmblk
: A GC_PARM_BLK containing the protocol connection status with the IPSET_MEDIA_STATE parameter set ID and one of the following parameter IDs:
  - IPPARM_TX_CONNECTED – Media streaming has been initiated in transmit direction. The value for this parameter ID is datatype IP_CAPABILITY and contains the coder configuration that resulted from the capability exchange with the remote peer.
  - IPPARM_TX_DISCONNECTED – Media streaming has been terminated in transmit direction. No parameter value is used with this parameter ID.
  - IPPARM_RX_CONNECTED – Media streaming has been initiated in receive direction. The value for this parameter ID is datatype IP_CAPABILITY and contains the coder configuration that resulted from the capability exchange with the remote peer.
  - IPPARM_RX_DISCONNECTED – Media streaming has been terminated in receive direction. No parameter value is used with this parameter ID.

4.9 Getting Notification of Underlying Protocol State Changes

The application can receive notification of intermediate protocol signaling state changes for both H.323 and SIP. The events for this notification must be enabled; see Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105.

Once these events are enabled, when a protocol state change occurs, the application receives a GCEV_EXTENSION event. The EXTENSIONEVTBLK structure pointed to by the extevtdatap pointer within the GCEV_EXTENSION event will contain the following information:

extID
: IPEXTID_IPPROTOCOL_STATE

parmblk

A GC_PARM_BLK containing the protocol connection status with the IPSET_IPPROTOCOL_STATE parameter set ID and one of the following parameter IDs:

- IPPARM_SIGNALING_CONNECTED – The signaling for the call has been established with the remote endpoint. For example, in H.323, a CONNECT message was received by the caller or a CONNECTACK message was received by the callee.
- IPPARM_SIGNALING_DISCONNECTED – The signaling for the call has been terminated with the remote endpoint. For example, in H.323, a RELEASE message was received by the terminator or a RELEASECOMPLETE message was received by peer.
- IPPARM_CONTROL_CONNECTED – Media control signaling for the call has been established with the remote endpoint. For example, in H.323, an OpenLogicalChannel message (for the receive direction) or an OpenLogicalCahnnelAck message (for the transmit direction) was received.
- IPPARM_CONTROL_DISCONNECTED – Media control signaling for the call has been terminated with the remote endpoint. For example, in H.323, an EndSession message was received.

Note: The parameter value field in this GC_PARM_BLK in each case is unused (NULL).

4.10 Sending Protocol Messages

The following message types are supported:

- Nonstandard User Input Indication (UII) Message (H.245)
- Nonstandard Facility Messages (Q.931)
- Nonstandard Registration Messages

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
Nonstandard UII Message (H.245)	IPSET_MSG_H245 • IPPARM_MSGTYPE (set to IP_MSGTYPE_H245_INDICATION)	Only when call is in Connected state
Nonstandard Facility Message (Q.931)	IPSET_MSG_Q931 • IPPARM_MSGTYPE (set to IP_MSGTYPE_Q931_FACILITY)	In any call state
Nonstandard Registration Message	IPSET_MSG_RAS • IPPARM_MSGTYPE (set to IP_MSGTYPE_REG_NONSTD	

4.10.1 Nonstandard UII Message (H.245)

To send nonstandard UII messages, use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. At the sending end, reception of a GCEV_EXTENSIONCMPLT event indicates that the message has been sent. At the receiving end, a GCEV_EXTENSION event with the same ext_id value is generated. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSION event is a pointer to an EXTENSIONEVTBLK structure which in turn contains a GC_PARM_BLK that includes 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_NONSTANDARDDATA
with either:
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_NONSTANDARDDATA_OBJID – Object ID string. The maximum length is MAX_NS_PARM_OBJID_LENGTH (40).

or
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_H221NONSTANDARD – H.221 nonstandard data identifier.

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

See Section 8.2.13, "IPSET_MSG_Q931", on page 225 and Section 8.2.16, "IPSET_NONSTANDARDDATA", on page 226 for more information.

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

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_NONSTANDARDDATA,
                             IPPARM_NONSTANDARDDATA_OBJID, ObjLen+1, ObjId);

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

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_ASYNC);
```

```
gc_util_delete_parm(t_PrmBlkp);
.
.
.
```

4.10.2 Nonstandard Facility Message (Q.931)

Use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG to send nonstandard facility (Q.931 Facility) messages. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. At the sending end, a GCEV_EXTENSIONCMPLT event is received indicating that the message has been sent. At the receiving end, a GCEV_EXTENSION event with the same ext_id value is generated. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSION event is a pointer to an EXTENSIONEVTBLK structure which in turn contains a GC_PARM_BLK that includes 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
with either:
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_NONSTANDARDDATA_OBJID – Object ID string. The maximum length is MAX_NS_PARM_OBJID_LENGTH (40).

or
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_H221NONSTANDARD – H.221 nonstandard data identifier.

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

See Section 8.2.13, “IPSET_MSG_Q931”, on page 225 and Section 8.2.16, “IPSET_NONSTANDARDDATA”, on page 226 for more information.

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

```
char ObjId[]= "1 22 333 4444";
char NSData[]= "DataField_Facility";

GC_PARM_BLKP    gcParmBlk = NULL;

gc_util_insert_parm_val(&gcParmBlk,
                         IPSET_MSG_Q931,
                         IPPARM_MSGTYPE,
                         sizeof(int),
                         IP_MSGTYPE_Q931_FACILITY);

gc_util_insert_parm_ref(&gcParmBlk,
                         IPSET_NONSTANDARDDATA,
                         IPPARM_NONSTANDARDDATA_OBJID,
                         sizeof(ObjId),
                         ObjId);
```

```
gc_util_insert_parm_ref(&gcParmBlk,
                         IPSET_NONSTANDARDDATA,
                         IPPARM_NONSTANDARDDATA_DATA,
                         sizeof(NSData),
                         NSData);

gc_Extension( GCTGT_GCLIB_CRN,
              crn,
              IPEXTID_SENDMSG,
              gcParmBlk,
              NULL,
              EV_ASYNC);

gc_util_delete_parm_blk(gcParmBlk);
```

4.10.3 Nonstandard Registration Message

Use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG to send nonstandard registration messages. The **target_type** should be GCTGT_CCLIB_NETIF and the **target_id** should be the board device handle, since the message destination is the Gatekeeper. At the sending end, a GCEV_EXTENSIONCMPLT event is received indicating that the message has been sent. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSION event is a pointer to an EXTENSIONEVTBLK structure, which in turn contains a GC_PARM_BLK that includes all of the data in the message.

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

IPSET_PROTOCOL
- IPPARM_PROTOCOL_BITMASK – Must be set to IP_PROTOCOL_H323

IPSET_MSG_REGISTRATION
- IPPARM_MSGTYPE – Set to IP_MSGTYPE_REG_NONSTD

IPSET_NONSTANDARDDATA
with either:
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_NONSTANDARDDATA_OBJID – Object ID string. The maximum length is MAX_NS_PARM_OBJID_LENGTH (40).

or
- IPPARM_NONSTANDARDDATA_DATA – Actual nonstandard data. The maximum length is MAX_NS_PARM_DATA_LENGTH (128).
- IPPARM_H221NONSTANDARD – H.221 nonstandard data identifier.

Note: The protocol and message type (IPPARM_MSGTYPE) are mandatory, and at least one other information element must be included.

The following code snippet illustrates how to send an H.221 nonstandard registration message.

```
{
    GC_PARM_BLKP parmblkp = NULL;
    char h221nonstd_id[] = "My H.221 Nonstandard data identifier"; /* must be
            <= MAX_NS_PARM_OBJID_LENGTH (40) */
    char nonstd_data[] = "My nonstandard_data";
```

```
        gc_util_insert_parm_val(&parmblkp, IPSET_PROTOCOL, IPPARM_PROTOCOL_BITMASK,
                  sizeof(char), IP_PROTOCOL_H323);
        gc_util_insert_parm_val(&parmblkp, IPSET_MSG_REGISTRATION, IPPARM_MSGTYPE,
                  sizeof(unsigned long), IP_MSGTYPE_REG_NONSTD);
        gc_util_insert_parm_ref(&parmblkp, IPSET_NONSTANDARDDATA, IPPARM_NONSTANDARDDATA_DATA,
                  sizeof(nonstd_data), nonstd_data);
        gc_util_insert_parm_ref(&parmblkp, IPSET_NONSTANDARDDATA, IPPARM_H221NONSTANDARD,
                  sizeof(h221nonstd_id), h221nonstd_id);

        if (gc_Extension(GCTGT_CCLIB_NETIF, bdev, IPEXTID_SENDMSG, parmblkp, NULL,
                  EV_ASYNC) != GC_SUCCESS)
        {
             printandlog(ALL_DEVICES, GC_APIERR, NULL, "gc_Extension() Failed", 0);
             exitdemo(1);
        }
}
```

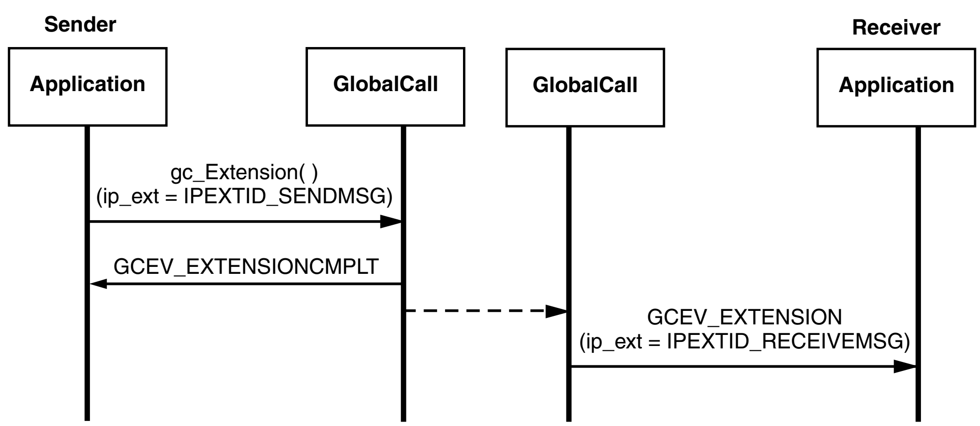
See Section 8.2.14, “IPSET_MSG_REGISTRATION”, on page 225 and Section 8.2.16, “IPSET_NONSTANDARDDATA”, on page 226 for more information.

4.10.4 Sending Facility, UII, or Registration Message Scenario

The **gc_Extension( )** function can be used to send H.245 UII messages or Q.931 nonstandard facility messages. Figure 28 shows this scenario.

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

Figure 28. Sending Protocol Messages



4.11 Enabling and Disabling Unsolicited Notification Events

The application can enable and disable the GCEV_EXTENSION events associated with unsolicited notification including:

- DTMF digit detection

- underlying protocol (Q.931 and H.245) connection state changes
- media streaming connection state changes
- T.38 fax events

Enabling and disabling unsolicited GCEV_EXTENSION notification events is done by manipulating the event mask, which has a default value of zero, using the **gc_SetConfigData( )** function. The relevant **gc_SetConfigData( )** function parameter values in this context are:

- **target_type** – GCTGT_CCLIB_NETIF
- **target_id** – IPT board device
- **size** – set to a value of GC_VALUE_LONG
- **target_datap –** a pointer to a GC_PARM_BLK structure that contains the parameters to be configured

The GC_PARM_BLK should contain the IPSET_EXTENSIONEVT_MSK parameter set ID and one of the following parameter IDs:

GCACT_ADDMSK
: Add an event to the mask

GCACT_SUBMSK
: Remove an event from the mask

GCACT_SETMSK
: Set the mask to a specific value

Possible values (corresponding to events that can be added or removed from the mask are) are:

EXTENSIONEVT_DTMF_ALPHANUMERIC
: For notification of DTMF digits received in User Input Indication (UII) messages with alphanumeric data. When using SIP, this value is not applicable.

EXTENSIONEVT_SIGNALING_STATUS
: For notification of intermediate protocol state changes in signaling (in H.323, for example, Q.931 Connected and Disconnected) and control (in H.323, for example, H.245 Connected and Disconnected).

EXTENSIONEVT_STREAMING_STATUS
: For notification of the status and configuration information of transmit or receive directions of media streaming including: Tx Connected, Tx Disconnected, Rx Connected, and Rx Disconnected.

EXTENSIONEVT_T38_STATUS
: For notification of fax tones detected on T.38 fax.

4.12 Configuring the Sending of the Proceeding Message

The application can configure if the Proceeding message is sent under application control (using the **gc_CallAck( )** function) or automatically by the stack. The **gc_SetConfigData( )** function can be used for this purpose.

The relevant set ID and parameter ID that must be included in the associated GC_PARM_BLK are:

GCSET_CHAN_CONFIG
GCPARM_CALLPROC. Possible values are:
- GCCONTROL_APP – The application must use **gc_CallAck( )** to send the Proceeding message. This is the default.
- GCCONTROL_TCCL – The stack sends the Proceeding message automatically.

4.13 Enabling and Disabling Tunneling in H.323

Tunneling is the encapsulation of H.245 media control messages within Q.931/H.225 signaling messages. If tunneling is enabled, one less TCP port is required for incoming connections.

For outgoing calls, the application can enable or disable tunneling by including the IPSET_CALLINFO parameter set ID and the IPPARM_H245TUNNELING parameter ID in the GCLIB_MAKECALL_BLK used by the **gc_MakeCall( )** function. Possible values for the IPPARM_H245TUNNELING parameter ID are:

- IP_H245TUNNELING_ON
- IP_H245TUNNELING_OFF

For incoming calls, tunneling is enabled by default, but it can be configured on a board device level (where a board device is a virtual entity that corresponds to a NIC or NIC address; see Section 2.3.2, “IPT Board Devices”, on page 35). This is done using the **gc_SetConfigData( )** function with target ID of the board device and the parameters above specified in the GC_PARM_BLKP structure associated with the **gc_SetConfigData( )** function.

Note: Tunneling for inbound calls can be configured on a board device basis only (using the **gc_SetConfigData( )** function). Tunneling for inbound calls **cannot** be configured on a per line device or per call basis (using the **gc_SetUserInfo( )** function).

4.14 Specifying RTP Stream Establishment

When using Global Call, RTP streaming can be established before the call is connected (that is, before the calling party receives the GCEV_CONNECTED event). This feature enables a voice message to be played to the calling party (for example, a message stating that the called party is unavailable for some reason) without the calling party being billed for the call.

The **gc_SetUserInfo( )** function can be used to specify call-related information such as coder information and display information before issuing **gc_CallAck( )**, **gc_AcceptCall( )** or

gc_AnswerCall( ). See Section 7.2.25, “gc_SetUserInfo( ) Variances for IP”, on page 198 for more information.

On the called party side, RTP streaming can be established before any of the following functions are issued to process the call:

- **gc_AcceptCall( )** – SIP Ringing (180) message returned to the calling party
- **gc_AnswerCall( )** – SIP OK (200) message returned to the calling party

4.15 Quality of Service Alarm Management

Global Call supports the setting and retrieving of Quality of Service (QoS) thresholds and the handling of a QoS alarm when it occurs. The QoS thresholds supported by Global Call are:

- lost packets
- jitter
- RTCP inactivity
- RTP inactivity

When using Global Call with other technologies (such as E1 CAS or T1 Robbed Bit), 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 rather than 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 API Programming Guide* for more information about GCAMS.

Note: Applications **must** include the *gcipmlib.h* header file before Global Call can be used to set or retrieve QoS threshold values.

4.15.1 Alarm Source Object Name

In Global Call, alarms are managed using the Global Call Alarm Management System (GCAMS). Each alarm source is represented by an Alarm Source Object (ASO) that has an associated name. When using Global Call with IP, the ASO name is **IPM QoS ASO**. The ASO name is useful in many contexts, for example, when configuring a device for alarm notification.

4.15.2 Retrieving the Media Device Handle

To retrieve the Global Call line device corresponding to the media device, use the **gc_GetResourceH( )** function. See Section 7.2.11, "gc_GetResourceH( ) Variances for IP", on page 169 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.

4.15.3 Setting QoS Threshold Values

To set QoS threshold values, use the **gc_SetAlarmParm( )** function. See Section 7.2.23, "gc_SetAlarmParm( ) Variances for IP", on page 194 for more information.

The following code demonstrates how to set QoS threshold values.

Note: The following code uses the IPM_QOS_THRESHOLD_INFO structure from the IP Media Library (IPML). See the *IP Media Library API Library Reference* and the *IP Media Library API Programming Guide* for more information.

```
/*****************************************************************************
Routine: SetAlarmParm
Assumptions/Warnings: None.
Description: calls gc_SetAlarmParm()
Parameters: handle of the Media device
Returns: None
*****************************************************************************/

void SetAlarmParm(int hMediaDevice)
{
    ALARM_PARM_LIST alarm_parm_list;
    IPM_QOS_THRESHOLD_INFO QoS_info;
    alarm_parm_list.n_parms = 1;
    QoS_info.unCount=1;
    QoS_info.QoSThresholdData[0].eQoSType = QOSTYPE_LOSTPACKETS;
    QoS_info.QoSThresholdData[0].unTimeInterval = 50;
    QoS_info.QoSThresholdData[0].unDebounceOn = 100;
    QoS_info.QoSThresholdData[0].unDebounceOff = 100;
    QoS_info.QoSThresholdData[0].unFaultThreshold = 10;
    QoS_info.QoSThresholdData[0].unPercentSuccessThreshold = 90;
    QoS_info.QoSThresholdData[0].unPercentFailThreshold = 10;

    alarm_parm_list.alarm_parm_fields[0].alarm_parm_data.pstruct =
    (void *) &QoS_info;

    if (gc_SetAlarmParm(hMediaDevice, ALARM_SOURCE_ID_NETWORK_ID,
          ParmSetID_qosthreshold_alarm, &alarm_parm_list, EV_SYNC)!= GC_SUCCESS)
    {
        /* handle gc_SetAlarmParm() failure */
        printf("SetAlarmParm(hMediaDevice=%d, mode=EV_SYNC) Failed", hMediaDevice);
        return;
    }
    printf("SetAlarmParm(hMediaDevice=%d, mode=EV_SYNC) Succeeded", hMediaDevice);
}
```

4.15.4 Retrieving QoS Threshold Values

To retrieve QoS threshold values, use the **gc_GetAlarmParm( )** function. See Section 7.2.8, "gc_GetAlarmParm( ) Variances for IP", on page 165 for more information.

The following code demonstrates how to retrieve QoS threshold values.

Note: The following code uses the IPM_QOS_THRESHOLD_INFO structure from the IP Media Library (IPML). See the *IP Media Library API Library Reference* and the *IP Media Library API Programming Guide* for more information.

```
/*******************************************************************************
Routine: GetAlarmParm
Assumptions/Warnings: None
Description: calls gc_GetAlarmParm()
Parameters: handle of Media device
Returns: None
********************************************************************************/

void GetAlarmParm(int hMediaDevice)
{
    ALARM_PARM_LIST alarm_parm_list;
    unsigned int n;
    IPM_QOS_THRESHOLD_INFO QoS_info;
    IPM_QOS_THRESHOLD_INFO *QoS_infop;

    QoS_info.unCount=2;
    QoS_info.QoSThresholdData[0].eQoSType = QOSTYPE_LOSTPACKETS;
    QoS_info.QoSThresholdData[1].eQoSType = QOSTYPE_JITTER;

    /* get QoS thresholds for LOSTPACKETS and JITTER */
    alarm_parm_list.alarm_parm_fields[0].alarm_parm_data.pstruct = (void *) &QoS_info;
    alarm_parm_list.n_parms = 1;

    if (gc_GetAlarmParm(hMediaDevice, ALARM_SOURCE_ID_NETWORK_ID,
          ParmSetID_qosthreshold_alarm, &alarm_parm_list, EV_SYNC) != GC_SUCCESS)
    {
        /* handle gc_GetAlarmParm() failure */
        printf("gc_GetAlarmParm(hMediaDevice=%d, mode=EV_SYNC) Failed", hMediaDevice);
        return;
    }

    /* display threshold values retrieved */
    printf("n_parms = %d\n", alarm_parm_list.n_parms);
    QoS_infop = alarm_parm_list.alarm_parm_fields[0].alarm_parm_data.pstruct;
    for (n=0; n < QoS_info.unCount; n++)
    {
        printf("QoS type = %d\n", QoS_infop->QoSThresholdData[n].eQoSType);
        printf("\tTime Interval = %u\n", QoS_infop->QoSThresholdData[n].unTimeInterval);
        printf("\tDebounce On = %u\n", QoS_infop->QoSThresholdData[n].unDebounceOn);
        printf("\tDebounce Off = %u\n", QoS_infop->QoSThresholdData[n].unDebounceOff);
        printf("\tFault Threshold = %u\n", QoS_infop->QoSThresholdData[n].unFaultThreshold);
        printf("\tPercent Success Threshold = %u\n",
               QoS_infop->QoSThresholdData[n].unPercentSuccessThreshold);
        printf("\tPercent Fail Threshold = %u\n",
               QoS_infop->QoSThresholdData[n].unPercentFailThreshold);
        printf("\n\n");
    }
}
```

4.15.5 Handling QoS Alarms

The application must first be enabled to receive notification of alarms on the specified line device. The following code demonstrates how this is achieved.

```
/*****************************************************************
*        NAME: enable_alarm_notification(struct channel *pline)
* DESCRIPTION: Enables all alarms notification for pline
*              Also fills in pline->mediah
*       INPUT: pline - pointer to channel data structure
*     RETURNS: None - exits if error
*    CAUTIONS: Does no sanity checking as to whether or not the technology
*              supports alarms - assumes caller has done that already
****************************************************************/

static void enable_alarm_notification(struct channel *pline)
{
    char     str[MAX_STRING_SIZE];
    int      alarm_ldev;                /* linedevice that alarms come on */

    alarm_ldev = pline->ldev;           /* until proven otherwise */

    if (pline->techtype == H323)
    {
        /* Recall that the alarms for IP come on the media device, not the network device */
        if (gc_GetResourceH(pline->ldev, &alarm_ldev, GC_MEDIADEVICE) != GC_SUCCESS)
        {
            sprintf(str, "gc_GetResourceH(linedev=%ld, &alarm_ldev,
                    GC_MEDIADEVICE) Failed", pline->ldev);
            printandlog(pline->index, GC_APIERR, NULL, str);
            exitdemo(1);
        }
        sprintf(str, "gc_GetResourceH(linedev=%ld, &alarm_ldev,
                GC_MEDIADEVICE) passed, mediah = %d", pline->ldev, alarm_ldev);
        printandlog(pline->index, MISC, NULL, str);
        pline->mediah = alarm_ldev;         /* save for later use */
    }
    else
    {
        printandlog(pline->index, MISC, NULL, "Not setting pline->mediah
                    since techtype != H323");
    }
    sprintf(str, "enable_alarm_notification - pline->mediah = %d\n", (int) pline->mediah);

    if (gc_SetAlarmNotifyAll(alarm_ldev, ALARM_SOURCE_ID_NETWORK_ID,
        ALARM_NOTIFY) != GC_SUCCESS)
    {
        sprintf(str, "gc_SetAlarmNotifyAll(linedev=%ld,
                ALARM_SOURCE_ID_NETWORK_ID, ALARM_NOTIFY) Failed", pline->ldev);
        printandlog(pline->index, GC_APIERR, NULL, str);
        exitdemo(1);
    }
    sprintf(str, "gc_SetAlarmNotifyAll(linedev=%ld, ALARM_SOURCE_ID_NETWORK_ID,
            ALARM_NOTIFY) PASSED", pline->ldev);
    printandlog(pline->index, MISC, NULL, str);
}
```

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. The following code demonstrates how to process a QoS alarm when it occurs. In this case the application simply logs information about the alarm.

```
/****************************************************************
*        NAME: void print_alarm_info(METAEVENTP metaeventp,
*                                    struct channel *pline)
* DESCRIPTION: Prints alarm information
*      INPUTS: metaeventp - pointer to the alarm event
*              pline - pointer to the channel data structure
*     RETURNS: NA
*    CAUTIONS: Assumes already known to be an alarm event
****************************************************************/

static void print_alarm_info(METAEVENTP metaeventp, struct channel *pline)
{
    long              alarm_number;
    char              *alarm_name;
    unsigned long     alarm_source_objectID;
    char              *alarm_source_object_name;
    char              str[MAX_STRING_SIZE];

    if (gc_AlarmNumber(metaeventp, &alarm_number) != GC_SUCCESS)
    {
        sprintf(str, "gc_AlarmNumber(...) FAILED");
        printandlog(pline->index, GC_APIERR, NULL, str);
        printandlog(pline->index, STATE, NULL, " ");
        exitdemo(1);
    }

    if (gc_AlarmName(metaeventp, &alarm_name) != GC_SUCCESS)
    {
        sprintf(str, "gc_AlarmName(...) FAILED");
        printandlog(pline->index, GC_APIERR, NULL, str);
        printandlog(pline->index, STATE, NULL, " ");
        exitdemo(1);
    }

    if (gc_AlarmSourceObjectID(metaeventp, &alarm_source_objectID) != GC_SUCCESS)
    {
        sprintf(str, "gc_AlarmSourceObjectID(...) FAILED");
        printandlog(pline->index, GC_APIERR, NULL, str);
        printandlog(pline->index, STATE, NULL, " ");
        exitdemo(1);
    }

    if (gc_AlarmSourceObjectName(metaeventp, &alarm_source_object_name) != GC_SUCCESS)
    {
        sprintf(str, "gc_AlarmSourceObjectName(...) FAILED");
        printandlog(pline->index, GC_APIERR, NULL, str);
        printandlog(pline->index, STATE, NULL, " ");
        exitdemo(1);
    }

    sprintf(str, "Alarm %s (%d) occurred on ASO %s (%d)",
            alarm_name, (int) alarm_number, alarm_source_object_name,
            (int) alarm_source_objectID);

    printandlog(pline->index, MISC, NULL, str);
}
```

See the *Global Call API Programming Guide* for more information about the operation of GCAMS and the *Global Call API Library Reference* for more information about GCAMS functions.

4.16 Registration

In an H.323 network, a gatekeeper manages the entities in a specific zone and an endpoint must register with the gatekeeper to become part of that zone. In a SIP network, a registrar performs a similar function. Global Call provides applications with the ability to perform endpoint registration. Registration tasks supported include:

- performing registration-related operations
- receiving notification of registration

When using Global Call to perform endpoint registration, the following restrictions apply:

- An application must use an IPT board device handle to perform registration. A board device handle can be obtained by using **gc_OpenEx( )** with a **devicename** parameter of “N_iptBx”.
- An application must perform registration before using **gc_OpenEx( )** on any other line device.
- Once an application chooses to be registered with a gatekeeper, it may change its gatekeeper/registrar by deregistering and reregistering with another gatekeeper/registrar, but it cannot handle calls without being registered with some gatekeeper/registrar.
- Once an application is registered, if it wishes to handle calls without the registration protocol (that is, return to the same mode as before registration), it can simply deregister.
- Once an application is registered and has active calls, deregistration or switching to a different gatekeeper must be done only when all calls are in the Idle state. The **gc_ResetLineDev( )** function can be used to put all channels in the Idle state.
- When setting alias information, if the protocol is H.323 only, the **gc_ReqService( )** function can include more than one alias in the GC_PARM_BLK associated with the function. If the registration target includes SIP, only one alias is supported and prefixes should not be included.
- When using the **gc_ReqService( )** function, two mandatory parameters IDs, PARM_REQTYPE and PARM_ACK, both in the GCSET_SERVREQ parameter set, are required in the GC_PARM_BLK parameter block. These parameters are required by the generic service request mechanism provided by Global Call and are not sent in any registration message.
- Registration operations cannot be included in the preset registration information using **gc_SetConfigData( )**.

4.16.1 Performing Registration Operations

Global Call provides a number of options for registration and manipulation of registration information. The Global Call API simplifies and abstracts the network RAS messages in H.323 and Registrar messages in SIP. The following functionality is supported:

- locating a registration server (gatekeeper in H.323 or registrar in SIP) via unicast or multicast (RAS messages: GRQ/GCF/GRJ)
- registration (RAS message: RRQ)
- specifying one-time or periodical registration (RAS message: RRQ)
- changing registered information (RAS message: RRQ)

- removing registered information by value (RAS message: RRQ)
- sending non-standard registration message (RAS message: NonStandardMessage)
- deregistering (RAS messages: URQ/UCF/URJ)
- handling calls according to the gatekeeper policy for directing and routing calls (RAS messages: ARQ/ACF/ARJ, DRQ/DCF/DRJ)

Note: For detailed information on RAS negotiation, see *ITU-T Recommendation H.225.0*.

SIP REGISTER

The SIP REGISTER method is used to register associations between a media endpoint alias and its real (transport) address. The associations are maintained in a SIP registrar and used for SIP call routing. Global Call supports only registering with a registrar, and does not support receiving SIP REGISTER methods. Table 11 associates abstract registrar registration concepts with SIP REGISTER elements and Global Call interface elements.

Table 11. SIP REGISTER Method

Concept	SIP REGISTER Element	Global Call Interface Element
Initiate registration	REGISTER method	gc_ReqService( )
Registrar's address	Request URI	IPSET_REG_INFO IP_REGISTER_ADDRESS.reg_server
Alias (Address-of-record)	To	IPSET_REG_INFO IP_REGISTER_ADDRESS.reg_client
Sender's address-of-record (alias) (same as address of record to be registered if registering own self)	From	None (this is OK)
Transport address (address bindings or real address, not alias)	Contact	IPSET_LOCAL_ALIAS (string)

Locating a Registration Server

A Global Call application can choose to use a known address for the registration server (gatekeeper in H.323 or registrar in SIP) or to discover a registration server by multicasting to a well-known address on which registration servers listen. This choice is determined by the IP address specified as the registration address during registration.

The registration address is specified in the IPPARM_REG_ADDRESS parameter in the IPSET_REG_INFO parameter set. The IPPARM_REG_ADDRESS is of type IP_REGISTER_ADDRESS, which contains the reg_server field that is the address value. A specific range of IP addresses is reserved for multicast transmission:

- If the application specifies an address in the range of multicast addresses or specifies the default multicast address (IP_REG_MULTICAST_DEFAULT_ADDR), then registration server discovery is selected.
- If the application specifies an address outside the range of multicast addresses, then registration with a specific server is selected.

Notes: ***1.*** The application can specify the maximum number of hops (connections between routers) in the max_hops field of the IP_REGISTER_ADDRESS structure. This field applies only to H.323 applications using gatekeeper discovery (H.225 RAS) via the default multicast registration address.

2. When using H.323, the port number used for RAS is one less than the port number used for signaling. Consequently, to avoid a conflict when configuring multiple devices in the IPCCLIB_START_DATA structure, do not assign consecutive H.323 signaling port numbers to devices. See Section 7.2.26, "gc_Start( ) Variances for IP", on page 201 for more information.

Registration

An application can use the **gc_ReqService( )** function to register with a gatekeeper/registrar. The registration information in this case is included in the GC_PARM_BLK associated with the **gc_ReqService( )** function. See Section 4.16.4, "Registration Code Example", on page 117 for more information.

If registration is initiated by a Global Call application via **gc_ReqService( )** and the gatekeeper rejects the registration, a GCEV_SERVICERESP event will be received with a reason of IPEC_RASReasonInvalidIPEC_RASAddress.

Specifying One-Time or Periodic Registration

Global Call enables an application to specify a one-time registration or periodic registration where information is re-registered with the gatekeeper/registrar at the interval (in seconds) specified by the application. This is achieved by setting the time_to_live field in the IP_REGISTER_ADDRESS structure. If the parameter is set to zero, then the stack uses one-time registration functionality. If the parameter is set to a value greater than zero, for example 5, then each registration with the server is valid for 5 seconds and the stack will automatically refresh its request before timeout. Registered applications are not notified of the refresh transactions.

When using SIP, periodic registration is also supported. The behavior depends on the time_to_live value specified in the IP_REGISTER_ADDRESS structure as follows:

- If the time_to_live value is specified, registration is done with this value set in the Expires header.
- If the time_to_live value is zero, the call control library automatically sets the Expires header to a value of 3600 seconds, which is treated as an application-specified time-to-live value.

Note: The actual expiration time for registration is determined by the registrar. The expiration time received from the registrar is stored and when half of this time expires, re-registration occurs.

If the gatekeeper rejects the registration (sends RRJ) during periodic registration, an unsolicited GCEV_TASKFAIL event will be received with a reason provided by the gatekeeper. If the gatekeeper does not set the reason, the reason is IPEC_RASReasonInvalidIPEC_RASAddress.

Changing Registration Information

Global Call provides the ability to modify or add to the registration information after it has been registered with the gatekeeper/registrar. To change registration information, use the **gc_ReqService( )** function. The GC_PARM_BLK in this context should contain an element with a set ID of IPSET_REG_INFO and a parameter ID of IPPARM_OPERATION_REGISTER that has a value of:

IP_REG_SET_INFO
: To override existing registration.

IP_REG_ADD_INFO
: To add to existing registration information.

The overriding or additional information is contained in other elements in the GC_PARM_BLK. The elements that can be included are given in Table 25, "Registration Information When Using H.323", on page 191 and Table 26, "Registration Information When Using SIP", on page 193.

Removing Registered Information by Value

When an application needs to delete one (or more) of its aliases or supported prefixes from the list, it may use the **gc_ReqService( )** function. The GC_PARM_BLK in this context should contain an element with a set ID of IPSET_REG_INFO and a parameter ID of IPPARM_OPERATION_REGISTER with a value of IP_REG_DELETE_BY_VALUE. If the string is registered, it will be deleted from the database and an updated list will be sent to the gatekeeper.

Note: When using IPPARM_OPERATION_REGISTER, the value IP_REG_DELETE_ALL is prohibited.

Sending Nonstandard Registration Messages

Global Call provides the ability to send nonstandard messages to and receive nonstandard messages from the gatekeeper or registrar. To send nonstandard messages, the application uses the **gc_Extension( )** function. The first element must be set as described in Section 8.2.14, "IPSET_MSG_REGISTRATION", on page 225. Other elements are set as in conventional nonstandard messages; see Section 8.2.16, "IPSET_NONSTANDARDDATA", on page 226.

Deregistering

Global Call provides the ability to deregister from a gatekeeper/registrar. To deregister, an application uses the **gc_ReqService( )** function. When deregistering, the application can decide whether to keep the registration information locally or delete it. The GC_PARM_BLK in this context should contain an element with a set ID of IPSET_REG_INFO and a parameter ID of IPPARM_OPERATION_DEREGISTER that has a value set to either:

IP_REG_MAINTAIN_LOCAL_INFO
: To keep the registration information locally.

IP_REG_DELETE_ALL
: To delete the registration information stored locally.

See Section 4.16.5, “Deregistration Code Example”, on page 119 for more information.

4.16.2 Receiving Notification of Registration

An application that sends a registration request to a gatekeeper/registrar will receive notification of whether the registration is successful or not. When using Global Call the application will receive a GCEV_SERVICERESP termination event with an associated GC_PARM_BLK that contains the following elements:

- IPSET_PROTOCOL parameter set ID with the IPPARM_PROTOCOL_BITMASK parameter ID that has one of the following values:
  - IP_PROTOCOL_H323
  - IP_PROTOCOL_SIP
- IPSET_REG_INFO parameter set ID with the IPPARM_REG_STATUS parameter ID that has one of the following values:
  - IP_REG_CONFIRMED
  - IP_REG_REJECTED

4.16.3 Receiving Nonstandard Registration Messages

An unsolicited GCEV_EXTENSION event with an extension ID (ext_id) of IPEXTID_RECEIVEMSG can be received that contains a nonstandard registration message. The associated GC_PARM_BLK contains the message details as follows:

- A message identifier element that contains the IPSET_MSG_REGISTRATION parameter set ID and an IPPARM_MSGTYPE parameter ID with a value of IP_MSGTYPE_REG_NONSTD.
- One or more additional elements that contain the message data of the form:
  - IPSET_NONSTANDARDDATA with
    - IPPARM_NONSTANDARDDATA_DATA – the maximum length is MAX_NS_PARM_DATA_LENGTH (128)
    - IPPARM_NONSTANDARDDATA_OBJID – the maximum length is MAX_NS_PARM_OBJID_LENGTH (40)

  OR
  - IPSET_NONSTANDARDDATA with
    - IPPARM_NONSTANDARDDATA_DATA – the maximum length is MAX_NS_PARM_DATA_LENGTH (128)
    - IPPARM_H221NONSTANDARD

4.16.4 Registration Code Example

The following code example shows how to populate a GC_PARM_DATA structure that can be used to register an endpoint with a gatekeeper (H.323) or registrar (SIP). The GC_PARM_DATA structure contains the following registration information:

- two mandatory parameters required by the generic **gc_ReqService( )** function

- the protocol type (H.323, SIP, or both)
- the type of operation (register/deregister) and sub-operation (set information, add information, delete by value, delete all)
- the IP address to be registered
- the endpoint type to register as
- a number of local aliases
- a number of supported prefixes

```
int boardRegistration(IN LINEDEV boarddev)
{
    GC_PARM_BLKP pParmBlock = NULL;
    int frc = GC_SUCCESS;

    /****** Two (mandatory) elements that are not related directly to
    the server-client negotiation ********/
    frc = gc_util_insert_parm_val(&pParmBlock,
                                  GCSET_SERVREQ,
                                  PARM_REQTYPE,
                                  sizeof(char),
                                  IP_REQTYPE_REGISTRATION);

    frc = gc_util_insert_parm_val(&pParmBlock,
                                  GCSET_SERVREQ,
                                  PARM_ACK,
                                  sizeof(char),
                                  1);

    /******Setting the protocol target***********/
    frc = gc_util_insert_parm_val(&pParmBlock,
                                  IPSET_PROTOCOL,
                                  IPPARM_PROTOCOL_BITMASK,
                                  sizeof(char),
                                  IP_PROTOCOL_H323); /*can be H323, SIP or Both*/

    /****** Setting the operation to perform ***********/
    frc = gc_util_insert_parm_val(&pParmBlock,
                                  IPSET_REG_INFO,
                                  IPPARM_OPERATION_REGISTER, /* can be Register or Deregister */
                                  sizeof(char),
                                  IP_REG_SET_INFO); /* can be other relevant "sub" operations */

    /****** Setting address information ***********/
    IP_REGISTER_ADDRESS registerAddress;
    strcpy(registerAddress.reg_server,"101.102.103.104"); /* set server address*/
    strcpy(registerAddress.reg_client,"user@10.20.30.40");   /* set alias for SIP*/
    registerAddress.max_hops = regMulticastHops;
    registerAddress.time_to_live = regTimeToLive;

    frc = gc_util_insert_parm_ref(&pParmBlock,
                                  IPSET_REG_INFO,
                                  IPPARM_REG_ADDRESS,
                                  (UINT8)sizeof(IP_REGISTER_ADDRESS),
                                  &registerAddress);

    /****** Setting endpoint type to GATEWAY (H.323 only) ***********/
    gc_util_insert_parm_ref(&pParmBlock,
                                  IPSET_REG_INFO,
                                  IPPARM_REG_TYPE,
                                  (unsigned char)sizeof(EPType),
                                  IP_REG_GATEWAY);
```

```
    /**** Setting terminalAlias information ****/
    /**** With H.323 - may repeat this line with different aliases and alias types ****/
    /**** SIP allows registering only a single transport address ****/
    frc = gc_util_insert_parm_ref(&pParmBlock,
                                  IPSET_LOCAL_ALIAS,
                                  (unsigned short)IPPARM_ADDRESS_EMAIL,
                                  (UINT8)(strlen("someone@someplace.com")+1),
                                  "someone@someplace.com");

    /****** Setting supportedPrefixes information ***********/
    /**** With H.323 - may repeat this line with different supported prefixes and
            supported prefix types ****/
    /**** SIP does not allow setting of this parm block ****/
    frc = gc_util_insert_parm_ref(&pParmBlock,
                                  IPSET_SUPPORTED_PREFIXES,
                                  (unsigned short)IPPARM_ADDRESS_PHONE,
                                  (UINT8)(strlen("011972")+1),
                                  "011972");

    /****** Send the request  ***********/
    unsigned long   serviceID ;
    int rc = gc_ReqService(GCTGT_CCLIB_NETIF,
                           boarddev,
                           &serviceID,
                           pParmBlock,
                           NULL,
                           EV_ASYNC);

    if (rc != GC_SUCCESS)
    {
        printf("failed in gc_ReqService\n");
        return GC_ERROR;
    }

    gc_util_delete_parm_blk(pParmBlock);
    return GC_SUCCESS;
}
```

4.16.5 Deregistration Code Example

The following code example shows how to populate a GC_PARM_DATA structure that can be used to deregister an endpoint with a gatekeeper (H.323). The GC_PARM_DATA structure contains the following deregistration information:

- the type of operation (in this case, deregister) and sub-operation (do not retain the registration information locally)
- two mandatory parameters required by the generic **gc_ReqService( )** function
- the protocol type (in this case, H.323)

```
void unregister()
{
    GC_PARM_BLKP        pParmBlock = NULL;
    unsigned long       serviceID = 1;
    int                 rc,frc;
    int gc_error;           // GC error code
    int cclibid;            // Call Control library ID for gc_ErrorValue
    long cc_error;          // Call Controll library error code
    char *resultmsg;        // String associated with cause code
    char *lib_name;         // Library name for cclibid
```

```
gc_util_insert_parm_val(&pParmBlock,
                        IPSET_REG_INFO,
                        IPPARM_OPERATION_DEREGISTER,
                        sizeof(unsigned char),
                        IP_REG_DELETE_ALL);

frc = gc_util_insert_parm_val(&pParmBlock,
                              GCSET_SERVREQ,
                              PARM_REQTYPE,
                              sizeof(unsigned char),
                              IP_REQTYPE_REGISTRATION);

if (frc != GC_SUCCESS)
{
    printf("failed in PARM_REQTYPE\n");
    termapp();
}

frc = gc_util_insert_parm_val(&pParmBlock,
                              GCSET_SERVREQ,
                              PARM_ACK,
                              sizeof(unsigned char),
                              IP_REQTYPE_REGISTRATION);

if (frc != GC_SUCCESS)
{
    printf("failed in PARM_ACK\n");
    termapp();
}

frc = gc_util_insert_parm_val(&pParmBlock,
                              IPSET_PROTOCOL,
                              IPPARM_PROTOCOL_BITMASK,
                              sizeof(char),
                              IP_PROTOCOL_H323); /*can be H323, SIP or Both*/

if (frc != GC_SUCCESS)
{
    printf("failed in IPSET_PROTOCOL\n");
    termapp();
}

rc = gc_ReqService(GCTGT_CCLIB_NETIF,
                   brddev,
                   &serviceID,
                   pParmBlock,
                   NULL,
                   EV_ASYNC);

if ( GC_SUCCESS != rc)
{
    printf("gc_ReqService failed while unregestering\n");
    if (gc_ErrorValue(&gc_error, &cclibid, &cc_error) != GC_SUCCESS)
    {
        printf("gc_Start() failed:  Unable to retrieve error value\n");
    }
    else
    {
        gc_ResultMsg(LIBID_GC, (long) gc_error, &resultmsg);
        printf("gc_ReqService() failed:  gc_error=0x%X:  %s\n", gc_error, resultmsg);
        gc_ResultMsg(cclibid, cc_error, &resultmsg);
        gc_CCLibIDToName(cclibid, &lib_name);
        printf("%s library had error 0x%lx - %s\n", lib_name, cc_error, resultmsg);
    }
    gc_util_delete_parm_blk(pParmBlock);
    exit(0);
}
```

```
    printf("Unregister request to the GK was sent ...\n");
    printf("the application will not be able to make calls !!! so it will EXIT\n");
    gc_util_delete_parm_blk(pParmBlock);
    return;
}
```

4.16.6 Gatekeeper Registration Failure

Gatekeeper registration may fail for any one of several reasons, such as disconnecting the network cable, a network topology change that result in the loss of all paths to the gatekeeper, a gatekeeper failure, or a gatekeeper shutdown. Terminals may not be immediately aware of the registration failure unless a RAS registration is attempted when the cable is disconnected, in which case the transaction fails immediately because of a socket bind failure. More typically, a RAS registration failure is only detected when either the Time To Live interval (programmable, with a default of 20 seconds) or the Response timeout (2 seconds) expires. RAS failure detection times can be improved by setting the Time To Live value in the RAS registration request to a value smaller than the default value, to 10 seconds, for example.

When RAS loses the gatekeeper registration, all existing calls are automatically disconnected by Global Call. All new calls are gracefully rejected and will continue to be rejected until RAS successfully registers or explicitly unregisters with the gatekeeper. The application can use the **gc_ReqService( )** function to perform the re-register or unregister operation. Calls in progress that are disconnected during RAS recovery are identified by a call control library result value of IPEC_RASReasonNotRegistered in the GCEV_DISCONNECTED event.

All **gc_ReqService( )** function calls result in the return of either a GCEV_SERVICERESP (success) or GCEV_TASKFAIL (fail) completion event. If RAS registration fails (e.g., as a result of an immediate socket bind failure or failure notification following a Time To Live timeout), the application receives a GCEV_TASKFAIL event. The range of applicable cause values for RAS-related GCEV_TASKFAIL events is IPEC_RASReasonMin to IPEC_RASReasonMax. The application must use the **gc_ReqService( )** function to reconfigure or register RAS in response to that event. If the RAS registration is rejected, the call control library is still cleaning up after the RAS registration failure and the application will receive another GCEV_TASKFAIL event, in which case it must issue **gc_ReqService( )** yet again.

It is recommended (but not required) that after receiving a GCEV_TASKFAIL event which identifies loss of gatekeeper registration, the application should:

- Stop attempting to make new calls, which uses resources unnecessarily and slows down the cleanup time.
- Immediately issue a new RAS register or RAS unregister request.

RAS registration requests should be made immediately on receipt of a RAS GCEV_TASKFAIL. Recovery from the loss of registration with the gatekeeper is not completed until the call control library re-registers or attempts to unregister. Re-registration or unregistration is not attempted by the call control library until commanded by the application using the **gc_ReqService( )** function to issue a RAS REGISTER REQUEST or a RAS UNREGISTER SERVICE REQUEST respectively.

4.17 Call Transfer When Using H.323

Global Call provides a common method of call transfer across technologies as described in the *Global Call API Programming Guide*. This section describes specific variances when using H.450.2 protocol (part of the H.323 protocol suite). For H.450.2-specific call scenarios see Section 3.2, "Call Transfer Scenarios When Using H.323", on page 42. The topics covered here include:

- Enabling Call Transfer
- Global Call Line Devices for Call Transfer
- Incoming Transferred Call
- Call Transfer Glare Condition

4.17.1 Enabling Call Transfer

Call transfer is a feature that can be disabled or enabled at the time the **gc_Start( )** function is called.

The **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** functions, which must be called before the **gc_Start( )** function, populate the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default values. The default value of the sup_serv_mask field in the IP_VIRTBOARD structure disables call transfer. The default sup_serv_mask field value must therefore be overridden with the value IP_SUP_SERV_CALL_XFER for each IPT board device on which call transfer is to be enabled. The following code snippet provides an example:

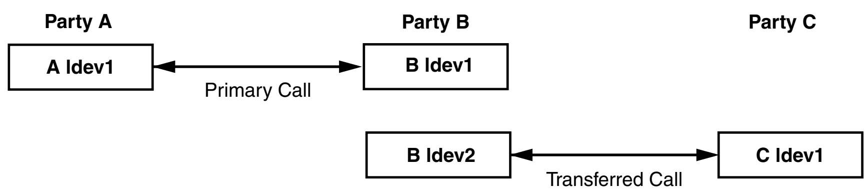
```
.
.
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].sup_serv_mask = IP_SUP_SERV_CALL_XFER; /* override supp services default */
ip_virtboard[1].sup_serv_mask = IP_SUP_SERV_CALL_XFER; /* override supp services default */
.
.
```

Note: If the application tries to use one of the call transfer functions, which include **gc_AcceptInitXfer( )**, **gc_AcceptXfer( )**, **gc_InitXfer( )**, **gc_InvokeXfer( )**, **gc_RejectInitXfer( )**, and **gc_RejectXfer( )**, and call transfer is not explicitly enabled via the IP_VIRTBOARD structure during **gc_Start( )**, the function call will fail, and the CCLIB error will be set to IPERR_SUP_SERV_DISABLED.

4.17.2 Global Call Line Devices for Call Transfer

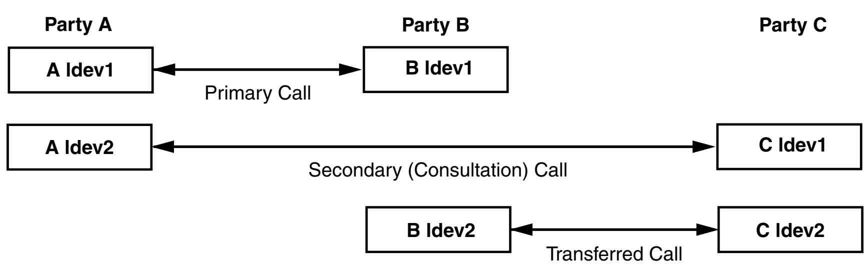
The Global Call IP architecture is designed so that each RTP transcoder at all times is streaming (xmit and rcv) with only one other endpoint. Thus, to support blind call transfer, two Global Call line devices are required at the transferred (party B) endpoint, one for the primary call with the transferring (party A) endpoint and a second to initiate the transferred call to the transferred-to (party C) endpoint. See Figure 29.

Figure 29. Global Call Devices for Blind Call Transfer



To support a successful supervised call transfer, two Global Call line devices are eventually utilized at all endpoints. The transferring (party A) endpoint makes a consultation call to the transferred-to (party C) endpoint, thus utilizing two line devices at both these endpoints as well. See Figure 30.

Figure 30. Global Call Devices for Supervised Call Transfer



4.17.3 Incoming Transferred Call

The incoming transferred call to party C contains the call control library (CCLIB) cause value of IPEC_IncomingTransfer and a Global Call library (GC LIB) cause value of GCRV_XFERCALL. The **gc_ResultInfo( )** function can be used to retrieve these values.

In the case of supervised transfer, the associated CRN of the secondary/consultation call is provided. The secondary CRN can be accessed via the extevtdatap pointer within the METAEVENT structure of the GCEV_OFFERED event which references a GC_PARM_BLK. From this parameter block, a data element identified by the SetId/ParmId pair of GCSET_SUPP_XFER and GCPARM_SECONDARYCALL_CRN can be retrieved via the parameter block utility functions to retrieve the secondary call CRN, which is of datatype size CRN (long).

If the transferee address is also provided to party C (optional for H.450.2), it can also be retrieved from this same parameter block, via a data element identified by the SetId/ParmId pair of GCSET_SUPP_XFER and GCPARM_TRANSFERRING_ADDR via the parameter block utility functions as a character array of maximum size GC_ADDRSIZE.

The following code sample demonstrates how to implement this:

```
   .
   .
   .
case GCEV_OFFERED:
{
```

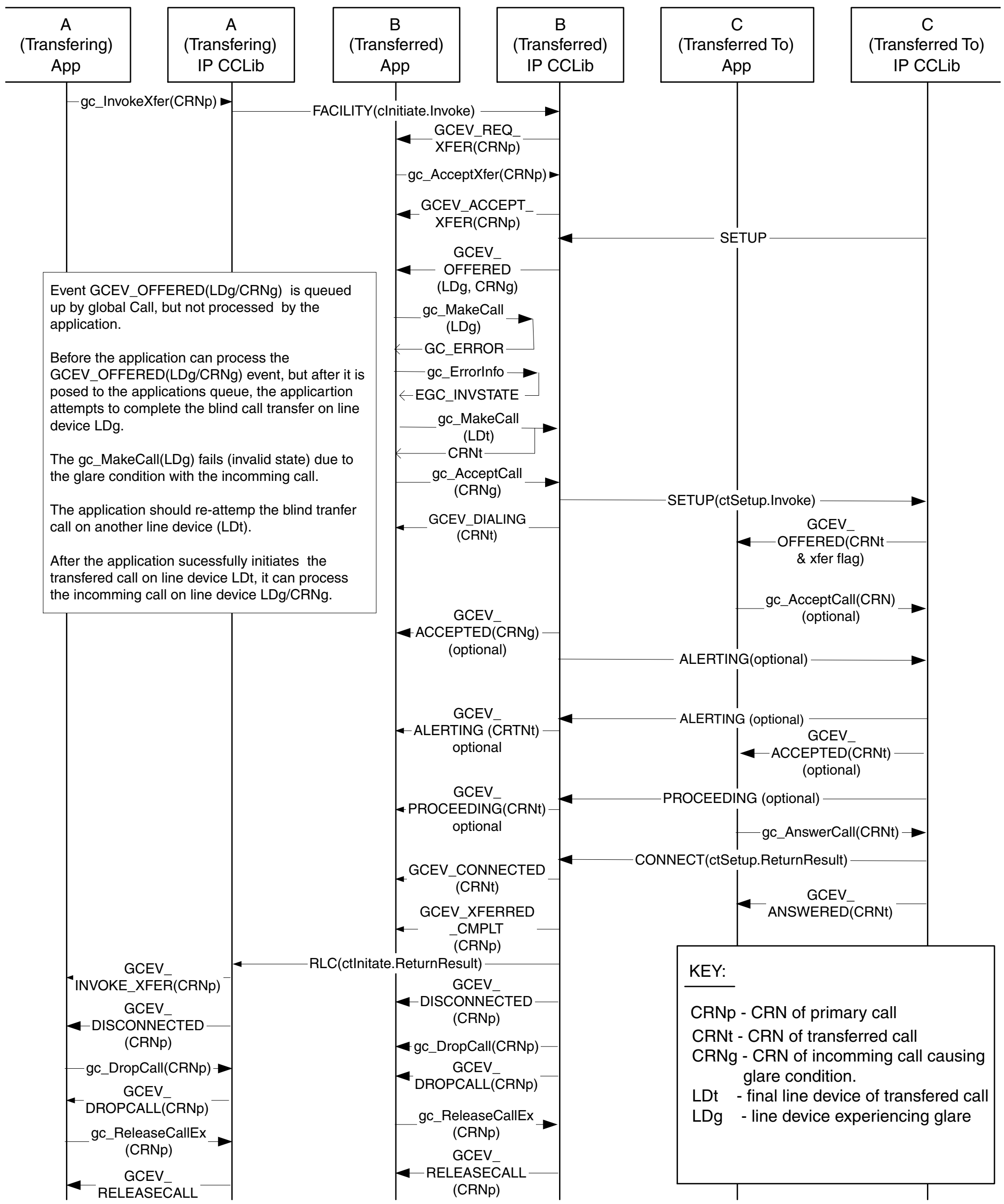
```
   if (metaevent.extevtdatap)
   {
      GC_PARM_BLKP parm_blkp = metaevent.extevtdatap;
      GC_PARM_DATAP curParm = NULL;
      printf("GCEV_OFFERED has parmblk:\n");
      while ((curParm = gc_util_next_parm(parm_blkp, curParm)) != NULL)
      {
         CRN secondaryCRN = 0;
         char transferringAddr[GC_ADDRSIZE];
         printf("SetID: 0x%x  ParmID: 0x%x\n",curParm->set_ID,curParm->parm_ID);

         switch (curParm->parm_ID)
         {
            case GCPARM_SECONDARYCALL_CRN:
               memcpy(&secondaryCRN, curParm->value_buf, curParm->value_size);
               printf("GCPARM_SECONDARYCALL_CRN: 0x%x\n",secondaryCRN);
               break;

            case GCPARM_TRANSFERRING_ADDR:
               memcpy(transferringAddr, curParm->value_buf, curParm->value_size);
               printf("GCPARM_TRANSFERRING_ADDR: %s\n",transferringAddr);
               break;

            default:
               printf("UNEXPECTED PARM_ID: %d\n",curParm->parm_ID);
               break;
         }
      }
   }
break;
.
.
.
```

4.17.4 Call Transfer Glare Condition

Glare may occur on a line device during both blind and supervised call transfer operations. Glare occurs on a line device during call transfer at Party B when the application calls **gc_MakeCall( )** to establish the transferred call (after the application has called **gc_AcceptXfer( )** on the primary CRN). Glare occurs because the CCLIB IP library has chosen the same line device for an incoming call that the application has chosen for establishing the transferred call. The application indication that this glare condition has occurred is that **gc_MakeCall( )** will fail with error EGC_INVSTATE, GCRV_GLARE or EGC_ILLSTATE. The application should retry the transferred call establishment request on another “available” line device. The application should process the GCEV_OFFERED metaevent on the incoming call/line device that caused the glare “normally” when it is retrieved. The call scenario in Figure 31 describes the glare condition and the appropriate application response.

Figure 31. Call Transfer Glare Condition

Precondition: Primary call between A and B is connected (not shown).



Post Condition: Transferred call between B and C Completed. Primary call between A and B is dropped and released. Incoming call that causes glare is ringing.

4.18 T.38 Fax Server Support

Global Call support for T.38 Fax Server is described under the following topics:

4.18.1 T.38 Fax Server Support Overview

Global Call provides T.38 fax server functionality to support fax-on-demand and other applications. The functionality includes the ability of an application to:

- Initiate and complete a T.38 session without an audio connection being first established.
- Switch coders from audio to T.38 fax and back again during a pre-established audio connection.

To support T.38 fax functionality, Global Call uses two types of media devices:

- A traditional Media device
- A new T.38 Fax device

By default, ipmBxCy represents the media device on board x and channel y, which has no fax capability on HMP. By associating the corresponding voice handle with a fax handle, the ipmBxCy device represents the fax channel defined by the fax handle, with no voice capability. Disassociating the voice and fax devices restores the ipmBxCy device voice capability.

Global Call uses the **gc_SetUserInfo( )** function to associate and disassociate a traditional Media device with a T.38 Fax device when establishing or concluding a T.38 fax connection. Manual device association must be done before the next Global Call function that requires fax information:

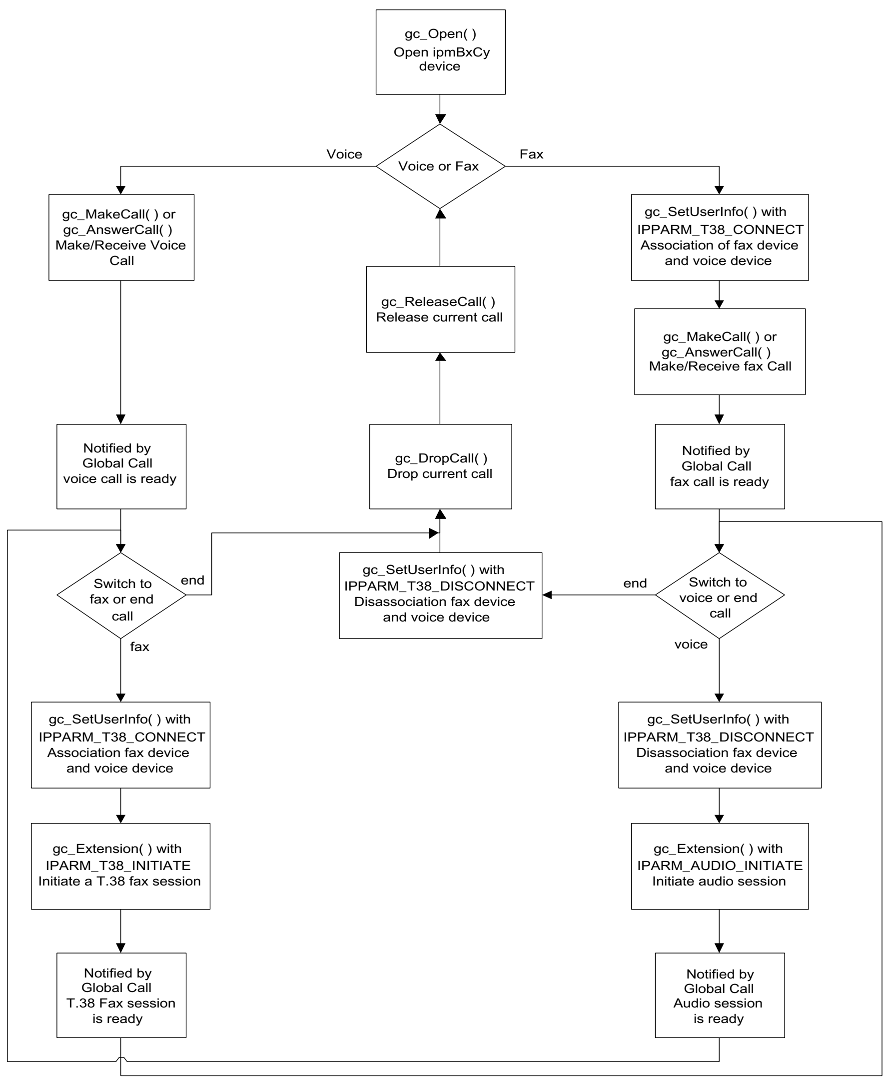
- For H.323, the association must be made before **gc_MakeCall( )** on the outbound call side, and **gc_CallAck( )**, **gc_AcceptCall( )** and **gc_AnswerCall( )** on the inbound call side, whichever occurs first since the underlying “open logical channel” can happen at any of these times if coder capabilities and fax port information is available.

- For SIP, the association must be made before **gc_MakeCall( )** on the outbound call side and **gc_AnswerCall( )** on the inbound call side, since media can only be opened after either of these functions.

Note: When a Media device is associated with a T.38 Fax device to establish a fax session over an existing audio connection, then when the fax session concludes, the Media device must be disassociated with the T.38 Fax device, **optionally** reestablishing the audio connection, **before** the call is dropped.

Figure 32 provides a flowchart that summarizes the T.38 fax server functionality and indicates the Global Call functions and events used at different stages in the call control process. The initial voice or fax capability decision before call connection is determined as described in Section 4.3.2.1, "Specifying Media Capabilities Before Connection", on page 72.

Figure 32. T.38 Fax Server Support in Manual Mode



4.18.2 Specifying Manual Operating Mode

An application must be configured in “Manual” mode to control the association and disassociation of Media and T.38 Fax devices during each call. The mode of operation is set on a board device basis. Once the GCEV_OPENEX event is received to confirm that the board device is open, the **gc_SetConfigData( )** function can be used to configure “Manual” mode as indicated in the code example below:

```
INT32 processEvtHandler()
{
   GC_PARM_BLK  *parmblkp = NULL;
   long         t = 0;
   :
   :
   switch (evtType)
   {
      :
      :
      case GCEV_OPENEX:
          gc_util_insert_parm_val(&parmblkp, IPSET_CONFIG,
                                  IPPARM_OPERATING_MODE, sizeof(int),
                                  IP_MANUAL_MODE);

          gc_SetConfigData(GCTGT_CCLIB_NETIF,
                           pline->ldev,
                           parmblkp,
                           0,
                           GCUPDATE_IMMEDIATE,
                           &t,
                           EV_ASYNC);

          gc_util_delete_parm_blk(parmblkp);
          break;
      :
      :
   }
   :
   :
}
```

4.18.3 Initiating a Switch from Audio to T.38 Fax

After an audio session has been established, the application can use the **gc_Extension( )** function to initiate a RequestMode (H.323) or Reinvite (SIP) to change the coder. Prior to initiating a coder change, the T.38 Fax device must be associated with the Media device. This can be achieved using the **gc_SetUserInfo( )** function. The application receives a GCEV_EXTENSION event to indicate that T.38 media is ready to send and receive fax information. The following code provides an example:

```
INT32 processEvtHandler()
{
   METAEVENT      metaEvent;
   GC_PARM_BLK    *parmblkp = NULL;
   IP_CONNECT     ipConnect;
     :
   switch (evtType)
   {
     :
      case GCEV_CONNECTED:
           /* received Connect event */
```

```
                /* in conversation */
                ipConnect.version = 0x100;
                ipConnect.mediaHandle = pline->mediaH;
                ipConnect.faxHandle = pline->faxH;
                ipConnect.connectType = IP_FULLDUP;
                gc_util_insert_parm_ref(&parmblkp, IPSET_FOIP,
                                        IPPARM_T38_CONNECT, (sizeof(IP_CONNECT)),
                                        (void *)(&ipConnect));

                gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn, parmblkp, GC_SINGLECALL);
                gc_util_delete_parm_blk(parmblkp);

                /* Initiate T.38 codec switch */
                gc_util_insert_parm_ref(&parmblkp, IPSET_SWITCH_CODEC,
                                        IPPARM_T38_INITIATE, sizeof(int), NULL);

                gc_Extension(GCTGT_GCLIB_CRN,pline->crn,
                             IPEXTID_CHANGEMODE, parmblkp, NULL, EV_ASYNC);

                gc_util_delete_parm_blk(parmblkp);

                }

                break;

             case GCEV_EXTENSIONCMPLT:
                   /* received extension complete event for T.38 initiation*/
                   /* do nothing */
                break;

             case GCEV_EXTENSION:
                   /* received extension event for media readiness */
                ext_evtblkp = (EXTENSIONEVTBLK *) metaEvent.extevtdatap;
                parmblkp = &ext_evtblkp->parmblk;

                while (t_gcParmDatap = gc_util_next_parm(parmblkp, t_gcParmDatap))
                {
                   switch(t_gcParmDatap->set_ID)
                   {
                      case IPSET_SWITCH_CODEC:
                         switch(t_gcParmDatap->parm_ID)
                         {
                            case IPPARM_READY:
                               /* Ready to send and receive fax */
                               fx_SndFax();
                               break;
                            :
                            :
                         }
                         break;
                   }
                }

                break;
            :
            }
         :
      }
```

4.18.4 Associating a T.38 Fax Device with a Media Device When a Fax Request is Received

During a voice call, a T.38 Fax request can be received by a RequestMode (H.323) or Reinvite (SIP) message. The application receives notification of the request as a GCEV_EXTENSION event. A T.38 Fax device must then be associated with the Media device by filling in an IP_CONNECT structure with the appropriate T.38 Fax and Media device handles and using the **gc_SetUserInfo( )** function. To continue to accept the request, the **gc_Extension( )** function is used as described in Section 4.18.5, "Accepting/Rejecting a Request to Switch from Audio to T.38 Fax or Vice Versa", on page 131. The following code provides an example:

```
INT32 processEvtHandler()
{
   METAEVENT        metaEvent;
   GC_PARM_BLK      *parmblkp = NULL;
   GC_PARM_DATAP    t_gcParmDatap = NULL;
   GC_PARM_BLK      *parmblkp2 = NULL;
   EXTENSIONEVTBLK  *ext_evtblkp = NULL;
   IP_CONNECT       ipConnect;
   :
   switch (evtType)
   {
      case GCEV_EXTENSION:
         /* received extension event, parse PARM_BLK examine
          * extension data
          */
         ext_evtblkp = (EXTENSIONEVTBLK *) metaEvent.extevtdatap;
         parmblkp = &ext_evtblkp->parmblk;
         while (t_gcParmDatap = gc_util_next_parm(parmblkp, t_gcParmDatap))
         {
            switch(t_gcParmDatap->set_ID)
            {
               case IPSET_SWITCH_CODEC:
                  switch(t_gcParmDatap->parm_ID)
                  {
                     case IPPARM_T38_REQUESTED:
                         /* connect the media and fax devices */
                         ipConnect.version = 0x100;
                         ipConnect.mediaHandle = pline->mediaH;
                         ipConnect.faxHandle = pline->faxH;
                         ipConnect.connectType = IP_FULLDUP;

                         gc_util_insert_parm_ref(&parmblkp2,
                                  IPSET_FOIP,
                                  IPPARM_T38_CONNECT, sizeof(IP_CONNECT),
                                  (void *)(&ipConnect));

                         gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn,
                                  parmblkp,GC_SINGLECALL);

                         gc_util_delete_parm_blk(parmblkp2);

                         /* accept T.38 request by example 4.17.5 */
                         acceptCodecSwitchRequest();
                         break;

                     case IPPARM_READY:
                        /* Ready to send and receive fax */
                        fx_SndFax();
                        break;
                  }
                  break;
            }
         }

   }
   :
}
```

4.18.5 Accepting/Rejecting a Request to Switch from Audio to T.38 Fax or Vice Versa

After T.38 coder change request has been received, followed by association of T.38 Fax device with Media device as described in Section 4.18.4, "Associating a T.38 Fax Device with a Media Device When a Fax Request is Received", on page 129, the application can use the **gc_Extension( )** function to accept or reject the request as follows:

- To accept the request, the GCPARM_BLK associated with the **gc_Extension( )** function includes components that indicate acceptance, specifically IPSET_SWITCH_CODEC and IPPARM_ACCEPT. A RequestModeAck (H.323) or 200 OK (SIP) message is not sent until the request is accepted. The following code provides an example:

```
/* Reject the incoming request */

INT32 acceptCodecSwitchRequest()
{
   GC_PARM_BLK *parmblkp = NULL;
   :
   gc_util_insert_parm_val(&parmblkp, IPSET_SWITCH_CODEC,IPPARM_ACCEPT, sizeof(int), NULL);

   gc_Extension(GCTGT_GCLIB_CRN,pline->crn,
                  PEXTID_CHANGEMODE, parmblkp, NULL, EV_ASYNC);

   gc_util_delete_parm_blk(parmblkp);

}
```

- To reject the request, the GCPARM_BLK associated with the **gc_Extension( )** function includes components that indicate rejection, specifically IPSET_SWITCH_CODEC and IPPARM_REJECT. The reason for rejecting the request is also included in the GCPARM_BLK. Chapter 10, "IP-Specific Event Cause Codes" describes the supported reject reasons that can be used in this context. For H.323, reasons prefixed by "IPEC_Q931Cause" can be used. For SIP, reasons prefixed by "IPEC_SIPReason" can be used. The reason parameter corresponds to a RequestModeReject cause (H.323) or a negative response code (SIP). The following code provides an example:

```
/* Reject the incoming request */

INT32 rejectCodecSwitchRequest()
{
   GC_PARM_BLK *parmblkp = NULL;
   :
   :
   /* Reject with reason being busy, SIP */
   gc_util_insert_parm_val(&parmblkp, IPSET_SWITCH_CODEC,IPPARM_REJECT,
                           sizeof(int), IPEC_SIPReasonStatus486BusyHere);

   gc_Extension(GCTGT_GCLIB_CRN,pline->crn, IPEXTID_CHANGEMODE, parmblkp, NULL, EV_ASYNC);

   gc_util_delete_parm_blk(parmblkp);
}
```

4.18.6 Sending a T.38 Fax in a Session Without Audio Established

Global Call supports the transmission of fax information in a session that does not already have an audio connection established. To send T.38 Fax in such a session, the application must use the **gc_SetConfigData( )** function to specify "Manual" mode, then use the **gc_SetUserInfo( )** function to associate a T.38 Fax device with a Media device before calling the **gc_MakeCall( )** function to actually send the fax information. The following code provides an example:

```
INT32 processEvtHandler()
{
   GC_PARM_BLK *parmblkp = NULL;
   :
   :
   switch (evtType)
   {
      case GCEV_OPENEX:
      /* Set manual mode */
      gc_util_insert_parm_val(&parmblkp, IPSET_CONFIG,
             IPPARM_OPERATING_MODE, sizeof(int), IP_MANUAL_MODE);

      gc_SetUserInfo(GCTGT_GCLIB_CHAN, pline->ldev, parmblkp, GC_ALLCALLS);

      gc_util_delete_parm_blk(parmblkp);

      /* Associate T.38 device with media device */
      ipConnect.version = 0x100;
      ipConnect.mediaHandle = pline->mediaH;
      ipConnect.faxHandle = pline->faxH;
      ipConnect.connectType = IP_FULLDUP;
      gc_util_insert_parm_ref(&parmblkp, IPSET_FOIP,
           IPPARM_T38_CONNECT, sizeof(IP_CONNECT),(void *)(&ipConnect));

      gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn, parmblkp,GC_SINGLECALL);
      gc_util_delete_parm_blk(parmblkp2);

      /* Make call now */
      gc_MakeCall();
      break;

      case GCEV_CONNECTED:
         Fx_sndFax();
         Break;
   }
   :
   :
}
```

4.18.7 Receiving a T.38 Fax in a Session Without Audio Established

Global Call supports the reception of fax information in a session that does not already have an audio connection established. The application can receive a GCEV_OFFERED event with a T.38 Fax request even if the session has no audio connection.

Note: The parameter block associated with the GCEV_OFFERED event indicates an incoming T.38 Fax request, if T.38 Fax is the **only** media offered in the incoming request. If more than T.38 media is offered, no specific T.38 information will be associated with offered event.

To answer the T.38 offer, the application must associate a Fax device with the Media device and set local T.38 media capability before calling the **gc_AnswerCall( )** function. The following code provides an example:

```
INT32 processEvtHandler()
{
   METAEVENT        metaEvent;
   GC_PARM_BLK      *parmblkp = NULL;
   GC_PARM_DATAP    t_gcParmDatap = NULL;
   GC_PARM_BLK      *parmblkp2 = NULL;
   EXTENSIONEVTBLK  *ext_evtblkp = NULL;
   IP_CONNECT       ipConnect;
   IP_CAPABILITY    ipcap;
   :
   switch (evtType)
   {
      case GCEV_OFFERED:
         /* parse PARM_BLK examine data */
         parmblkp = (GC_PARM_BLK *)metaEvent.extevtdatap;

         while (t_gcParmDatap = gc_util_next_parm(parmblkp, t_gcParmDatap))
         {
            switch(t_gcParmDatap->set_ID)
            {
               case IPSET_SWITCH_CODEC:
                  switch(t_gcParmDatap->parm_ID)
                  {
                     case IPPARM_T38_OFFERED:
                         /* connect media with fax devices */
                         ipConnect.version = 0x100;
                         ipConnect.mediaHandle = pline->mediaH;
                         ipConnect.faxHandle = pline->faxH;
                         ipConnect.connectType = IP_FULLDUP;

                         gc_util_insert_parm_ref(&parmblkp2, IPSET_FOIP,
                                IPPARM_T38_CONNECT, (sizeof(IP_CONNECT)),
                                (void *)(&ipConnect));
                         gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn,
                                parmblkp2,GC_SINGLECALL);

                         gc_util_delete_parm_blk(parmblkp2);

                         /* set T.38 media capability*/
                         ipcap.capability = GCCAP_DATA_t38UDPFax;
                         ipcap.type = GCCAPTYPE_RDATA;
                         ipcap.direction = IP_CAP_DIR_LCLTXRX;
                         ipcap.extra.data.max_bit_rate = 144;

                         gc_util_insert_parm_ref(&parmblkp2,
                                    GCSET_CHAN_CAPABILITY,
                                    IPPARM_LOCAL_CAPABILITY,
                                    sizeof(IP_CAPABILITY),
                                    &ipcap);

                         gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn,
                                    pParmBlock2, GC_SINGLECALL);

                         gc_util_delete_parm_blk(pParmBlock2);


                         /* received completion event for gc_Extension() */
                         gc_AnswerCall(pline->crn, 0, EV_ASYNC);
                         break;
```

```
                  }
               }
            }
         }
      }
```

4.18.8 Sending a Request to Switch from T.38 Fax to Audio

To request a switch from a T.38 Fax session back to an audio session, the application uses the **gc_Extension( )** function, which initiates a RequestMode (H.323) or Reinvite (SIP) message to actually perform the action. Before initiating the change of coder, the Fax device must be disassociated from the Media device using the **gc_SetUserInfo( )** function. The application receives a GCEV_EXTENSION event to indicate that audio can now be sent and received. The following code provides and example:

```
INT32 switchFromFaxToAudio()
{
   GC_PARM_BLK       *parmblkp = NULL;
   IP_CONNECT        ipConnect;

   ipConnect.version = 0x100;
   ipConnect.mediaHandle = pline->mediaH;

   gc_util_insert_parm_ref(&parmblkp, IPSET_FOIP, IPPARM_T38_DISCONNECT,
                           (sizeof(IP_CONNECT)), (void *)(&ipConnect));

   gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn, parmblkp, GC_SINGLECALL);

   gc_util_delete_parm_blk(parmblkp);

   /* Initiate audio codec switch */
   gc_util_insert_parm_ref(&parmblkp, IPSET_SWITCH_CODEC,
                            IPPARM_AUDIO_INITIATE, sizeof(int), NULL);

   gc_Extension(GCTGT_GCLIB_CRN,pline->crn, IPEXTID_CHANGEMODE, parmblkp, NULL, EV_ASYNC);

   gc_util_delete_parm_blk(parmblkp);

}

INT32 processEvtHandler()
{
   METAEVENT       metaEvent;
   GC_PARM_BLK     *parmblkp = NULL;
   :
   switch (evtType)
   {
      case GCEV_EXTENSIONCMPLT:
         /* received extension complete event for audio initiation*/
         /* do nothing */
         break;

      case GCEV_EXTENSION:
         /* received extension event for media readiness */
         ext_evtblkp = (EXTENSIONEVTBLK *) metaEvent.extevtdatap;
         parmblkp = &ext_evtblkp->parmblk;

         while (t_gcParmDatap = gc_util_next_parm(parmblkp, t_gcParmDatap))
         {
            switch(t_gcParmDatap->set_ID)
            {
               case IPSET_SWITCH_CODEC:
                  switch(t_gcParmDatap->parm_ID)
```

```
                  {
                     case IPPARM_READY:
                        /* Ready to send and receive audio */
                        gc_Listen();
                        break;
                     :
                     :
                  }
                  break;
            }
         }
         break;
      :
   }
 :
}
```

4.18.9 Receiving a Request to Switch from T.38 Fax to Audio

An application may receive a request to switch from a T.38 Fax session back to an audio session. The request is received as a GCEV_EXTENSION event that is triggered by a RequestMode (H.323) or Reinvite (SIP) message. Before accepting the incoming request, the application must disassociate the T.38 Fax device from the Media device using the **gc_SetUserInfo( )** function, then continue accepting the request as described in Section 4.18.5, "Accepting/Rejecting a Request to Switch from Audio to T.38 Fax or Vice Versa", on page 131.

```
INT32 processEvtHandler()
{
   METAEVENT        metaEvent;
   GC_PARM_BLK      *parmblkp = NULL;
   GC_PARM_DATAP    t_gcParmDatap = NULL;
   GC_PARM_BLK      *parmblkp2 = NULL;
   EXTENSIONEVTBLK  *ext_evtblkp = NULL;
   IP_CONNECT       ipConnect;
   :
   switch (evtType)
   {
      case GCEV_EXTENSION:
         /* received extension event, parse PARM_BLK examine
          * extension data
          */
         ext_evtblkp = (EXTENSIONEVTBLK *) metaEvent.extevtdatap;
         parmblkp = &ext_evtblkp->parmblk;
         while (t_gcParmDatap = gc_util_next_parm(parmblkp, t_gcParmDatap))
         {
            switch(t_gcParmDatap->set_ID)
            {
               case IPSET_SWITCH_CODEC:
                  switch(t_gcParmDatap->parm_ID)
                  {
                     case IPPARM_AUDIO_REQUESTED:
                        /* disconnect the media and fax devices */
                        ipConnect.version = 0x100;
                        ipConnect.mediaHandle = pline->mediaH;

                        gc_util_insert_parm_ref(&parmblkp2, IPSET_FOIP,
                                      IPPARM_T38_DISCONNECT, sizeof(IP_CONNECT),
                                      (void *)(&ipConnect));
```

```
                        gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn,
                                      parmblkp,GC_SINGLECALL);

                        gc_util_delete_parm_blk(parmblkp2);

                        /* accept audio request by example 4.3.3 */
                        acceptCodecSwitchRequest();
                        break;

                     case IPPARM_READY:
                        /* Ready to send and receive audio */
                        gc_Listen();
                        break;
                  }
                  break;

            }
         }

   }
  :
}
```

4.18.10 Terminating a Call After a T.38 Fax Session

After a T.38 fax session is finished, and prior to issuing **gc_DropCall( )**, the T.38 Fax device needs to be disassociated from the Media device using the **gc_SetUserInfo( )** function. The following code provides and example.

```
INT32 processEvtHandler()
{
METAEVENT      metaEvent;
GC_PARM_BLK    *parmblkp = NULL;
IP_CONNECT     ipConnect;
   :
   switch (evtType)
      {
         case GCEV_DISCONNECTED:
            /* received extension event, parse PARM_BLK examine
             * extension data
             */

            /* disconnect the media and fax devices */
            ipConnect.version = 0x100;
            ipConnect.mediaHandle = pline->mediaH;

            gc_util_insert_parm_ref(&parmblkp, IPSET_FOIP,
                                    IPPARM_T38_DISCONNECT, sizeof(IP_CONNECT),
                                    (void *)(&ipConnect));

            gc_SetUserInfo(GCTGT_GCLIB_CRN, pline->crn, parmblkp,GC_SINGLECALL);

            gc_util_delete_parm_blk(parmblkp);

            /* dropcall */
            gc_DropCall(pline->crn,GC_NORMAL_CLEARING, EV_ASYNC);
            break;
      }
   :
}
```

4.19 Using Object Identifiers

Object Identifiers (OIDs) are not free strings, they are standardized and assigned by various controlling authorities such as, the International Telecommunications Union (ITU), British Standards Institute (BSI), American National Standards Institute (ANSI), Internet Assigned Numbers Authority (IANA), International Standards Organization (ISO), and public corporations. Depending on the authority, OIDs use different encoding and decoding schemes. Vendors, companies, governments and others may purchase one or more OIDs to use while communicating with another entity on the network. For more information about OIDs, see *http://www.alvestrand.no/objectid/*.

An application may want to convey an OID to the remote side. This can be achieved by setting the OID string in any nonstandard parameter that can be sent in any Setup, Proceeding, Alerting, Connect, Facility, or User Input Indication (UII) message.

Global Call supports the use of any valid OID by allowing the OID string to be included in the GC_PARM_BLK associated with the specific message using the relevant parameter set ID and parameter IDs. Global Call will not send an OID that is not in a valid format. For more information on the valid OID formats see *http://asn-1.com/x660.htm* which defines the general procedures for the operation of OSI (Open System Interconnection) registration authorities.

The application is responsible for the validity and legality of any OID used.

Building Global Call IP Applications 5

This chapter describes the IP-specific header files and libraries required when building applications.

Note: For more information about building applications, see the *Global Call API Programming Guide*.

5.1 Header Files

When compiling Global Call applications for the IP technology, it is necessary to include the following header files in addition to the standard Global Call header files, which are listed in the *Global Call API Library Reference* and *Global Call API Programming Guide*:

gcip.h
: IP-specific data structures

gcip_defs.h
: IP-specific type definitions, error codes and IP-specific parameter set IDs and parameter IDs

gccfgparm.h
: Global Call type definitions, configurable parameters in the Global Call library and generic parameter set IDs and parameter IDs

gcipmlib.h
: for Quality of Service (QoS) features

5.2 Required Libraries

When building Global Call applications for the IP technology, it is not necessary to link any libraries other than the standard Global Call library, *libgc.lib*. Other libraries, including IP-specific libraries, are loaded automatically.

5.3 Required System Software

The Intel® NetStructure™ Host Media Processing software must be installed on the development system. See the Software Installation Guide for your HMP release for further information.

Debugging Global Call IP Applications 6

This chapter provides information about debugging Global Call IP applications:

6.1 Debugging Overview

The Global Call software can be configured to write underlying call control library and stack information to log files while an application is running. This information can help trace the sequence of events and identify the source of a problem. These log files are also useful when reporting problems to technical support personnel.

Table 12 shows the log files that can be generated, the directory into which each log file is written, the purpose of each log file, and the configuration files that can be used to customize the output each log file.

Note: Log file generation is enabled by placing the configuration file in the respective directory as indicated in Table 12.

Table 12. Summary of Log File Options

Log File Name	Purpose	Where Generated	Configuration File	Placement of Configuration File for Log Generation
gc_h3r.log	Call control library and SIP stack debugging	Application directory	*gc_h3r.cfg*	Application directory
logfile.log	H.323 stack debugging on Linux operating systems	Application directory	*msg.conf*	Application directory
rvtsp1.log	H.323 stack debugging in a Windows environment	Application directory	*rvtele.ini*	WINNT directory; for example, *C:\Winnt\rvtele.ini*

6.2 Log Files

The following topics provide information about the use of each log file:

- Call Control Library and SIP Stack Debugging
- H.323 Stack Debugging on Linux Operating Systems
- H.323 Stack Debugging

6.2.1 Call Control Library and SIP Stack Debugging

The *gc_h3r.cfg* file can be used to customize the information written to the *gc_h3r.log* file by one or both of the following:

- Customizing Call Control Library Logging to the gc_h3r.log File
- Customizing SIP Stack Logging to the gc_h3r.log File

6.2.1.1 Customizing Call Control Library Logging to the gc_h3r.log File

The call control library comprises three library files; *libgch3r.so*, *libsigal.so*, and *libsipsigal.so* for Linux operating systems, and *libgch3r.dll*, *libsigal.dll*, and *libsipsigal.dll* for Windows operating systems.

The modules in *libgch3r.so* and *libgch3r.dll* and the type of log information generated by each module are:

M_CRN
: Call Reference Number information and states.

M_SHM
: Interface to User information (for example, when calling **gc_ReleaseCallEx**( ), ShmReleaseCallEx is output to the log).

M_LD
: Line device operation and states.

M_MEDIA
: Media channel related information.

M_PDL
: Predefined Library information.

M_PACKER
: Packed event information to the application.

M_SH_DB
: Preconfiguration information saved in the control library.

M_SH_DEC
: Internal process communication information.

M_SH_ENC
: Internal process communication information.

M_SH_IPC
: Internal process communication information.

M_SH_UNPACK
: Information unpacked from the application.

M_BOARD
: Board-related information.

The modules in *libsigal.so* and *libsigal.dll* and the type of log information generated by each module are:

M_SIG_MAN
: DLL manager information.

M_CALL_MAN
: Call manager information.

M_SIGNAL
: Q.931 manager information.

M_CONTROL
: H.245 manager information.

M_CHAN_MAN
: Logical channel manager information.

M_CHAN
: Logical channel information.

M_IE
: Information element information.

M_SIG_DEC
: Internal process communication information.

M_SIG_ENC
: Internal process communication information.

M_SIG_IPC
: Internal process communication information.

M_RAS
: Registration, Admission and Status information.

M_CAPS
: Capability matching algorithm information.

The modules in *libsipsigal.so* and *libsipsigal.dll* and the type of log information generated by each module are:

M_S_SIGAL
: DLL manager information.

M_S_CALLM
: Call manager information.

M_S_SIGNL
: SIP manager information.

M_S_CHMGR
: Logical channel manager information.

M_SIP_IE
: Information element information.

M_SIP_CAP
: Capability matching algorithm information.

M_SIP_DEC
Internal process communication information.

M_SIP_ENC
Internal process communication information.

M_SIP_IPC
Internal process communication information.

M_INFO
Message send information (not yet implemented).

M_REFER
Call transfer information (not yet implemented).

M_PRACK
Provisional response information (not yet implemented).

M_AUTHENT
Authentication information (not yet implemented).

In the *gc_h3r.cfg* file, you can set a different debug level for each module. Table 13 shows the valid debug levels.

Table 13. Levels of Debug for Call Control Library Logging

Debug Level†	Debug Information	Call Control Library Output to Log File
0	L_NONE	No information
1	L_SPECIAL	Limited information describing call control library configuration
2	L_ERROR	Error information. This is the default level.
3	L_WARNING	Warning information
4	L_INFO	Significant state transition information
5	L_EXTEND	Additional relevant information
6	L_ALL	All information
†Selecting a debug level automatically includes all lower debug levels. For example, selecting level 3 automatically includes levels 0, 1, and 2.		

To set a module to the desired debug level, use the following syntax:

```
Module name = Debug Level Number
```

Some examples are:

`m_sip_enc = 2`
sets the m_sip_enc module to the LEVEL_ERROR debug level

`m_call_man = 6`
sets the m_call_man module to the LEVEL_ALL debug level

`m_media = 4`
sets the m_media module to the LEVEL_INFO debug level

6.2.1.2 Customizing SIP Stack Logging to the gc_h3r.log File

The SIP stack comprises a number of modules as follows:

- RvSipStack_Message
- RvSipStack_Core
- RvSipStack_Transport
- RvSipStack_Transaction
- RvSipStack_Call
- RvSipStack_Parser
- RvSipStack_Stack
- RvSipStack_Authenticator
- RvSipStack_RegClient
- RvSipStack_MsgBuilder

In the *gc_h3r.cfg* file, you can set different debug levels for each module. Table 14 shows the valid debug levels that can be set. More than one level of debug can be set for each module. This is achieved by specifying a decimal number that is the binary equivalent of the binary values of each desired level ORed together. For example, a value of 31 decimal (11111 binary) enables all debug levels.

Table 14. Levels of Debug Information for SIP Stack Logging

Debug Level †	Debug Information	SIP Stack Output to Log File
0	None	No information. This is the default level.
1	DEBUG	Detailed information about SIP stack activity.
2	INFO	Information about SIP stack activity.
4	WARNING	Warnings about possible non-fatal errors.
8	ERROR	A non-fatal error occurred.
16	EXCEP	A fatal error occurred that blocks stack operation.
† Uses a decimal representation of a bit mask, that is level 2 is 010, level 8 is 01000, level 16 is 010000 etc.		

Special gc_h3r.cfg Configuration Parameters

In the *gc_h3r.cfg* file, two parameters that have special significance are:

outputdest
: This parameter should be always be set to 0.

print_file_n_line
: Controls whether filename and line numbers are written to the log. Possible values are:
  - 1 – Filenames and line numbers are written to the log file.
  - 0 – Filenames and line numbers are not written to the log file.

Sample Extract from gc_h3r.log File

The following is an extract from a *gc_h3r.log* file:

```
4 ! 09:35:54.558 ! M_MEDIA   ! L_INFO    !    2 ! >> eventHandler:
ev CNTRL_EV_RX_CONNECT st TRL_ST_TXCONNECT
4 ! 09:35:54.558 ! M_MEDIA   ! L_INFO    !    2 ! >> Media::connectALL
4 ! 09:35:54.558 ! M_MEDIA   ! L_INFO    !    2 ! >> mediaEventHandler:
ev EV_CONNECTALL st ST_WAIT_FOR_CALL
6 ! 09:35:54.558 ! M_MEDIA   ! L_ALL     !    2 ! >> MediaState::startTransaction
6 ! 09:35:54.558 ! M_MEDIA   ! L_ALL     !    2 ! >> Mediaipml::setEventMaskCALL : mode
32768,maskEvt 0x7f
6 ! 09:35:54.558 ! M_MEDIA   ! L_ALL     !    2 ! << Mediaipml::setEventMaskCALL
:m_EventMaskState 0x22 : [0]
6 ! 09:35:54.558 ! M_MEDIA   ! L_ALL     !    2 ! >> Mediaipml::setRemoteMediaInfoCALL
5 ! 09:35:54.558 ! M_MEDIA   ! L_EXTEND  !    2 ! LOCAL_RTP Port = 2720 Address : 10.242.212.44
5 ! 09:35:54.568 ! M_MEDIA   ! L_EXTEND  !    2 ! LOCAL_RTCP Port = 2721 Address : 10.242.212.44
5 ! 09:35:54.568 ! M_MEDIA   ! L_EXTEND  !    2 ! REMOTE_RTP  Port = 2710 Address :
10.242.212.44
5 ! 09:35:54.568 ! M_MEDIA   ! L_EXTEND  !    2 ! LOCAL_CODER, G711ULAW64K,FSize 20,FPP 1,V 0,
PT 0,RedPT 0
5 ! 09:35:54.568 ! M_MEDIA   ! L_EXTEND  !    2 ! REMOTE_RTCP Port = 2711 Address :
10.242.212.44
5 ! 09:35:54.568 ! M_MEDIA   ! L_EXTEND  !    2 ! REMOTE_CODER, G711ULAW64K,FSize 20,FPP 1,V
0,PT 0,RedPT 0
6 ! 09:35:54.568 ! M_MEDIA   ! L_ALL     !    2 ! << Mediaipml::setRemoteMediaInfoCALL: [0]
6 ! 09:35:54.568 ! M_MEDIA   ! L_ALL     !    2 ! << MediaState::startTransaction: [0]
4 ! 09:35:54.568 ! M_MEDIA   ! L_INFO    !    2 ! << mediaEventHandler: ev EV_CONNECTALL st
ST_STARTING: [0
```

Notes: **1.** Lines that begin with 4 (level 4) indicate state machine transitions.

2. Lines that begin with 5 (level 5) provide extended information (in this case the remote coder and the local coder and RTP/RTCP information starting the media channel).

3. Lines that begin with 6 (level 6) provide additional information (in this case the entry to and exit from functions).

6.2.2 H.323 Stack Debugging on Linux Operating Systems

The *msg.conf* file can be used to customize the logging of H.323 stack information to the *logfile.log* file. You can use the *msg.conf* file for the following:

- Selecting Modules that Write to logfile.log
- Selecting the Debug Level
- Selecting the Debug Output Type

6.2.2.1 Selecting Modules that Write to logfile.log

The H.323 stack comprises a number of modules as follows:

Note: † indicates the most commonly used modules.

- EMA
- MEMORY
- RA
- CAT

- CM†
- CMAPI†
- CMAPICB†
- CMERR†
- TPKTCHAN†
- CONFIG†
- APPL
- FASTSTART†
- VT
- UNREG
- RAS†
- UDPCHAN
- TCP
- TRANSPORT
- ETIMER
- PER†
- PERERR†
- TUNNCTRL†
- Q931†
- Q931ERR
- L1
- TIMER
- AnnexE
- SSEERR
- SSEAPI
- SSEAPICS
- SSCHAN
- SUPS

In the *msg.conf* file, you can enable or disable the modules that write information to *logfile.log*. A module is disabled by including a pound symbol (#) in front of the module name. For example, in the following segment of the *msg.conf* file, the TPKTCHAN and UDPCHAN modules write to the log file, but the CMAPICB and CMAPI modules do not.

```
TPKTCHAN
UDPCHAN
#CMAPICB
#CMAPI
```

6.2.2.2 Selecting the Debug Level

In the *msg.conf* file, you can set the debug level by including lines similar to the following:

```
#set up debug level
%2
```

In this example, the debug level is set to 2. The valid debug levels and their meanings are:

0
: Do not display or print debug information. This is the default level.

1
: Do not display trees or do not check trees for ASN.1 consistency.

2
: Display all information including trees, but do not check trees for ASN.1 consistency.

3
: Display all information and check trees for ASN.1 consistency. Display any inconsistencies found.

4
: Display all messages.

6.2.2.3 Selecting the Debug Output Type

In the *msg.conf* file, you can direct where the debug output will be written by including lines similar to the following:

```
#debug output definition
>file
```

In this example, the debug output is directed to a file. The valid output options are:

file
: Write the debug output to a file. The name of the file is *<current directory>logfile.log*.

logger
: Write the debug output to a logger, such as the debug logger, or to any other printing tool.

terminal
: Display the debug output on a terminal.

6.2.3 H.323 Stack Debugging

The *rvtele.ini* file can be used to customize the logging of H.323 stack information to the *rvtsp1.log* file. You can use the *rvtele.ini* file for the following:

- Selecting Modules that Write to rvtsp1.log
- Selecting the Debug Level
- Selecting the Debug Output Type
- Configuring Cyclic Mode Parameters

6.2.3.1 Selecting Modules that Write to rvtsp1.log

The H.323 stack comprises a number of modules as follows:

Note: † indicates the most commonly used modules.

- EMA
- MEMORY
- RA
- CAT
- CM†
- CMAPI†
- CMAPICB†
- CMERR†
- TPKTCHAN†
- CONFIG†
- APPL
- FASTSTART†
- VT
- UNREG
- RAS†
- UDPCHAN
- TCP
- TRANSPORT
- ETIMER
- PER†
- PERERR†
- TUNNCTRL†
- Q931†
- Q931ERR
- L1
- TIMER
- AnnexE
- SSEERR
- SSEAPI
- SSEAPICS
- SSCHAN
- SUPS

In the *rvtele.ini* file, you can enable or disable modules from writing information to *rvtsp1.log*. A module is enabled by including a line with the <module name>=1 under a section labeled [insertIntoFile]. For example:

```
[insertIntoFile]
TPKTCHAN=1
UDPCHAN=1
CMPAPICB=0
CMPAPI=0
```

In this example, the TPKTCHAN and UDPCHAN modules write to the log file, but the CMAPICB and CMAPI modules do not.

Note: Only one section labeled [insertIntoFile] is allowed in the *rvtele.ini* file.

6.2.3.2 Selecting the Debug Level

In the *rvtele.ini* file, you can set the debug level by including lines similar to the following:

```
#set up debug level
deblevel=2
```

In this example, the debug level is set to 2. The valid debug levels and their meanings are:

0
: Do not display or print debug information. This is the default level.

1
: Display messages from all source modules, except those source modules in the list given with the filtering level instructions.

2
: Display messages from all source modules according to the list given with the filtering level instructions.

3
: Display messages from all source modules.

6.2.3.3 Selecting the Debug Output Type

In the *rvtele.ini* file, the following section must exist to direct the debug output:

```
[supserve]
...
msgfile=1
msgdeb=1
msgwin=0
```

In this example, the debug output is directed to the *rvtsp1.log* file and writes all debug output to the debugger window, such as the Windows debugger when running the application in a Windows environment. The valid output options are:

msgfile
: The stack writes all debug messages to the *rvtsp1.log* file.

msgdeb
: The stack writes all debug messages to a debugger window.

msgwin

The stack writes all debug messages to a special window that it creates.

6.2.3.4 Configuring Cyclic Mode Parameters

When using a debug output of msgfile=1, it is possible to work in cyclic mode and set a limit to the physical size of the *rvtsp1.log* file. When the log file expands to this size, information will be logged to the beginning of the file overwriting older logging information. The lines in the *rvtele.ini* file that control these parameters are as follows:

```
[fileParams]
fileSize=20000000
fileCyclic=1
```

To disable this option, set the parameter values as follows:

```
[fileParams]
fileSize=-1
fileCyclic=0
```

IP-Specific Function Information 7

Certain Global Call functions have additional functionality or perform differently when used with IP technology. The generic function descriptions in the *Global Call API Library 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 Library Reference* to obtain the complete information when developing Global Call applications that use IP technology. IP-specific variances are described in the following topics:

7.1 Global Call Functions Supported by IP

The following is a full list of the Global Call functions that indicates the level of support when used with IP technology. The list indicates whether the function is supported, not supported, or supported with variances.

gc_AcceptCall( )
: Supported with variances described in Section 7.2.1, "gc_AcceptCall( ) Variances for IP", on page 160

gc_AcceptInitXfer( )
: Supported

gc_AcceptXfer( )
: Supported

gc_AcceptInitXfer( )
: Supported with variances described in Section 7.2.2, "gc_AcceptInitXfer( ) Variances for IP", on page 160

gc_AcceptXfer( )
: Supported with variances described in Section 7.2.3, "gc_AcceptXfer( ) Variances for IP", on page 161

gc_AlarmName( )
: Supported

gc_AlarmNumber( )
: Supported

gc_AlarmNumberToName( )
Supported

gc_AlarmSourceObjectID( )
Supported

gc_AlarmSourceObjectIDToName( )
Supported

gc_AlarmSourceObjectName( )
Supported

gc_AlarmSourceObjectNameToID( )
Supported

gc_AnswerCall( )
Supported with variances described in Section 7.2.4, "gc_AnswerCall( ) Variances for IP", on page 162

gc_Attach( )
Not supported

gc_AttachResource( )
Supported

gc_BlindTransfer( )
Not supported

gc_CallAck( )
Supported with variances described in Section 7.2.5, "gc_CallAck( ) Variances for IP", on page 163

gc_CallProgress( )
Not supported

gc_CCLibIDToName( )
Supported

gc_CCLibNameToID( )
Supported

gc_CCLibStatus( )
Supported, but deprecated. Use **gc_CCLibStatusEx( )**.

gc_CCLibStatusAll( )
Supported, but deprecated. Use **gc_CCLibStatusEx( )**.

gc_CCLibStatusEx( )
Supported

gc_Close( )
Supported

gc_CompleteTransfer( )
Not supported

gc_CRN2LineDev( )
Supported

gc_Detach( )
Supported

gc_DropCall( )
Supported with variances described in Section 7.2.6, "gc_DropCall( ) Variances for IP", on page 163

gc_ErrorInfo( )
Supported

gc_ErrorValue( )
Supported, but deprecated. Use **gc_ErrorInfo( )**.

gc_Extension( )
Supported with variances described in Section 7.2.7, "gc_Extension( ) Variances for IP", on page 164

gc_GetAlarmConfiguration( )
Supported

gc_GetAlarmFlow( )
Supported

gc_GetAlarmParm( )
Supported with variances described in Section 7.2.8, "gc_GetAlarmParm( ) Variances for IP", on page 165

gc_GetAlarmSourceObjectList( )
Supported

gc_GetAlarmSourceObjectNetworkID( )
Supported

gc_GetANI( )
Not supported

gc_GetBilling( )
Not supported

gc_GetCallInfo( )
Supported with variances described in Section 7.2.9, "gc_GetCallInfo( ) Variances for IP", on page 166

gc_GetCallProgressParm( )
Not supported

gc_GetCallState( )
Supported

gc_GetConfigData( )
Not supported

gc_GetCRN( )
Supported

gc_GetCTInfo( )
Supported with variances described in Section 7.2.10, "gc_GetCTInfo( ) Variances for IP", on page 169

gc_GetDNIS( )
Not supported

gc_GetFrame( )
Not supported

gc_GetInfoElem( )
Not supported

gc_GetLineDev( )
Supported

gc_GetLineDevState( )
Not supported

gc_GetMetaEvent( )
Supported.

gc_GetMetaEventEx( )
Supported (Windows extended asynchronous mode only)

gc_GetNetCRV( )
Not supported

gc_GetNetworkH( )
Not supported

gc_GetParm( )
Not supported

gc_GetResourceH( )
Supported with variances described in Section 7.2.11, "gc_GetResourceH( ) Variances for IP", on page 169

gc_GetSigInfo( )
Not supported

gc_GetUserInfo( )
Not supported

gc_GetUsrAttr( )
Supported

gc_GetVer( )
Supported

gc_GetVoiceH( )
Not supported

gc_GetXmitSlot( )
Supported with variances described in Section 7.2.12, "gc_GetXmitSlot( ) Variances for IP", on page 170

gc_HoldACK( )
Not supported

gc_HoldCall( )
Not supported

gc_HoldRej( )
Not supported

gc_InitXfer( )
Supported

gc_InvokeXfer( )
Supported

gc_InitXfer( )
Supported with variances described in Section 7.2.13, “gc_InitXfer( ) Variances for IP”, on page 170

gc_InvokeXfer( )
Supported with variances described in Section 7.2.14, “gc_InvokeXfer( ) Variances for IP”, on page 170

gc_LinedevToCCLIBID( )
Supported

gc_Listen( )
Supported with variances described in Section 7.2.15, “gc_Listen( ) Variances for IP”, on page 172

gc_LoadDxParm( )
Not supported

gc_MakeCall( )
Supported with variances described in Section 7.2.16, “gc_MakeCall( ) Variances for IP”, on page 172

gc_Open( )
Not supported

gc_OpenEx( )
Supported with variances described in Section 7.2.17, “gc_OpenEx( ) Variances for IP”, on page 187

gc_QueryConfigData( )
Not supported

gc_RejectInitXfer( )
Supported

gc_RejectXfer( )
Supported

gc_RejectInitXfer( )
Supported with variances described in Section 7.2.18, “gc_RejectInitXfer( ) Variances for IP”, on page 189

gc_RejectXfer( )
Supported with variances described in Section 7.2.19, “gc_RejectXfer( ) Variances for IP”, on page 189

gc_ReleaseCall( )
Not supported

gc_ReleaseCallEx( )
Supported with variances described in Section 7.2.20, "gc_ReleaseCallEx( ) Variances for IP", on page 190

gc_ReqANI( )
Not supported

gc_ReqMoreInfo( )
Not supported

gc_ReqService( )
Supported with variances described in Section 7.2.21, "gc_ReqService( ) Variances for IP", on page 190

gc_ResetLineDev( )
Supported

gc_RespService( )
Supported with variances described in Section 7.2.22, "gc_RespService( ) Variances for IP", on page 194

gc_ResultInfo( )
Supported

gc_ResultMsg( )
Not supported

gc_ResultValue( )
Not supported

gc_RetrieveAck( )
Not supported

gc_RetrieveCall( )
Not supported

gc_RetrieveRej( )
Not supported

gc_SendMoreInfo( )
Not supported

gc_SetAlarmConfiguration( )
Supported

gc_SetAlarmFlow( )
Supported

gc_SetAlarmNotifyAll( )
Supported

gc_SetAlarmParm( )
Supported with variances described in Section 7.2.23, "gc_SetAlarmParm( ) Variances for IP", on page 194

gc_SetBilling( )
Not supported

gc_SetCallingNum( )
Not supported

gc_SetCallProgressParm( )
Not supported

gc_SetChanState( )
Not supported

gc_SetConfigData( )
Supported with variances described in Section 7.2.24, “gc_SetConfigData( ) Variances for IP”, on page 195

gc_SetEvtMask( )
Not supported

gc_SetInfoElem( )
Not supported

gc_SetParm( )
Not supported

gc_SetUpTransfer( )
Not supported

gc_SetUserInfo( )
Supported with variances described in Section 7.2.25, “gc_SetUserInfo( ) Variances for IP”, on page 198

gc_SetUsrAttr( )
Supported

gc_SndFrame( )
Not supported

gc_SndMsg( )
Not supported

gc_Start( )
Supported with variances described in Section 7.2.26, “gc_Start( ) Variances for IP”, on page 201

gc_StartTrace( )
Not supported

gc_Stop( )
Supported

gc_StopTrace( )
Not supported

gc_StopTransmitAlarms( )
Not supported

gc_SwapHold( )
Not supported

gc_TransmitAlarms( )
Not supported

gc_UnListen( )
Supported with variances described in Section 7.2.27, “gc_UnListen( ) Variances for IP”, on page 203

gc_util_delete_parm_blk( )
Supported

gc_util_find_parm( )
Supported

gc_util_insert_parm_ref( )
Supported

gc_util_insert_parm_val( )
Supported

gc_util_next_parm( )
Supported

gc_WaitCall( )
Supported

7.2 Global Call Function Variances for IP

Note: All functions are supported in asynchronous mode. Functions that also support synchronous mode (**gc_OpenEx**( ), **gc_Listen**( ), **gc_ReleaseCallEx**( ), and **gc_Unlisten**( )) are noted explicitly.

The Global Call function variances that apply when using IP technology are described in the following sections. See the *Global Call API Library Reference* for generic (technology-independent) descriptions of the Global Call API functions.

7.2.1 gc_AcceptCall( ) Variances for IP

The **rings** parameter is ignored.

Variance for H.323

The **gc_AcceptCall( )** function is used to send the Q.931 ALERTING message to the originating endpoint.

Variance for SIP

The **gc_AcceptCall**( ) function is used to send the 180 Ringing message to the originating endpoint.

7.2.2 gc_AcceptInitXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

Either the rerouting_num (of type char*) or rerouting_addrblkp (of type GCLIB_ADDRESS_BLK*) fields of the GC_REROUTING_INFO structure can be used to specify the rerouting address string to be signaled back to party A and its final destination to party B. The sub_address fields of the GCLIB_ADDRESS_BLK are ignored and not used.

Note: If both fields are used, the rerouting address string will be a concatenation of the information from both fields.

The GCEV_ACCEPT_INIT_XFER event is received by the application on the secondary/consultation call CRN once the transferred call is received as notified via the GCEV_OFFERED event.

If the call transfer is abandoned by parties A or B before the transfer is completed, the GCEV_ACCEPT_INIT_XFER_FAIL event is received with a CCLIB cause value of IPEC_H4502CTAbandon and a Global Call cause value of GCRV_CALLABANDONED.

If the CTT2 timer (20 seconds) expires before the transfer is completed, the GCEV_ACCEPT_INIT_XFER_FAIL event is received with a CCLIB cause value of IPEC_H450CTT2Timeout and a Global Call cause value of GCRV_TIMEOUT.

7.2.3 gc_AcceptXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

The **parmblkp** parameter is ignored for IP technology and should be set to NULL.

The **gc_AcceptXfer( )** function can be used at party B only on the receipt of GCEV_REQ_XFER.

Both the rerouting_num (of type char *) and the rerouting_addr (of type GCLIB_ADDRESS_BLK) fields of the GC_REROUTING_INFO structure contain the same rerouting address string as explicitly signaled from party A in blind call transfer, or party C in supervised call transfer via **gc_AcceptInitXfer( )**. The rerouting number to be used in the subsequent **gc_MakeCall( )** at party B can be copied from either element, but not a concatenation of both as they contain the same character string.

The remaining elements of the GCLIB_ADDRESS_BLK structure dereferenced from rerouting_addr contain the following:

address_type
: GCADDRTYPE_IP

address_plan
: GCADDRPLAN_UNKNOWN

sub_address
: 0 (unused)

sub_address_type
: 0 (unused)

sub_address_plan
: 0 (unused)

7.2.4 gc_AnswerCall( ) Variances for IP

The **rings** parameter is ignored.

Coders can be set in advance of using **gc_AnswerCall**( ) by using **gc_SetUserInfo( )**. See Section 7.2.25, "gc_SetUserInfo( ) Variances for IP", on page 198 for more information.

The following code example shows how to use the **gc_SetUserInfo( )** function to set coder information before calls are answered using **gc_AnswerCall**( ).

```
/* Specifying coders before answering calls */
LINEDEV ldev;
CRN crn;
GC_PARM_BLK *target_datap;
/* Define Coder */
IP_CAPABILITY a_DefaultCapability;
gc_OpenEx(&ldev, ":N_iptB1T1:M_ipmB1C1:P_H323", EV_ASYNC, 0);

/* wait for GCEV_OPENEX event ... */

/* Set default coder for this ldev */
target_datap = NULL;
memset(&a_DefaultCapability,0,sizeof(IP_CAPABILITY));
a_DefaultCapability.capability = GCCAP_AUDIO_g7231_5_3k;
a_DefaultCapability.direction = IP_CAP_DIR_LCLTRANSMIT;
a_DefaultCapability.type = GCCAPTYPE_AUDIO;
a_DefaultCapability.extra.audio.frames_per_pkt = 1;
a_DefaultCapability.extra.audio.VAD = GCPV_DISABLE;
gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY),
&a_DefaultCapability);

/* set both receive and transmit coders to be the same (since
   the IPTxxx board does not support asymmetrical coders */
memset(&a_DefaultCapability,0,sizeof(IP_CAPABILITY));
a_DefaultCapability.capability = GCCAP_AUDIO_g7231_5_3k;
a_DefaultCapability.direction = IP_CAP_DIR_LCLRECEIVE;
a_DefaultCapability.type = GCCAPTYPE_AUDIO;
a_DefaultCapability.extra.audio.frames_per_pkt = 1;
a_DefaultCapability.extra.audio.VAD = GCPV_DISABLE;
gc_util_insert_parm_ref(&target_datap, GCSET_CHAN_CAPABILITY,
IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY),
&a_DefaultCapability);

gc_SetUserInfo(GCTGT_GCLIB_CHAN, ldev, target_datap, GC_ALLCALLS);
gc_util_delete_parm_blk(target_datap);
gc_WaitCall(ldev, NULL, NULL, 0, EV_ASYNC);

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

/*... Retrieve crn from metaevent... */

gc_AnswerCall(crn, 0, EV_ASYNC);

/*... Receive GCEV_ANSWERED ... */
```

Variance for H.323

The **gc_AnswerCall**( ) function is used to send the Q.931 CONNECT message to the originating endpoint.

Variance for SIP

The **gc_AnswerCall( )** function is used to send the 200 OK message to the originating endpoint.

7.2.5 gc_CallAck( ) Variances for IP

The **callack_blkp** parameter must be a pointer to a GC_CALLACK_BLK structure that contains a type field with a value of GCACK_SERVICE_PROC. The following code example shows how to set up a GC_CALLACK_BLK structure and issue the **gc_CallAck( )** function.

```
GC_CALLACK_BLK gcCallAckBlk;
memset(&gcCallAckBlk, 0, sizeof(GC_CALLACK_BLK));
gcCallAckBlk.type = GCACK_SERVICE_PROC;
rc = gc_CallAck(crn, &gcCallAckBlk, EV_ASYNC);
```

The application can configure whether the Proceeding message is sent manually using the **gc_CallAck( )** function or whether it is sent automatically by the stack. See Section 4.12, “Configuring the Sending of the Proceeding Message”, on page 107 for more information.

Variance for H.323

The **gc_CallAck( )** function is used to send the Proceeding message to the originating endpoint.

Variance for SIP

The **gc_CallAck( )** function is used to send the 100 Trying message to the originating endpoint.

7.2.6 gc_DropCall( ) Variances for IP

The **cause** parameter can be any of the generic cause codes documented in the **gc_DropCall( )** function reference page in the *Global Call API Library Reference* or a protocol-specific cause code as described below.

Variance for H.323

Allowable protocol-specific cause codes are prefixed by IPEC_H225 or IPEC_Q931 in Chapter 10, “IP-Specific Event Cause Codes”.

Variance for SIP

Allowable protocol-specific cause codes are prefixed by IPEC_SIP in Chapter 10, “IP-Specific Event Cause Codes”.

Note: Cause codes and reasons are only supported when **gc_DropCall( )** is issued while the call is in the Offered state.

7.2.7 gc_Extension( ) Variances for IP

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

- retrieving call-related information
- getting notification of underlying protocol connection or disconnection state transitions
- getting notification of media streaming initiation and termination in both the transmit and receive directions
- specifying which DTMF types, when detected, provide notification to the application
- sending DTMF digits
- retrieving protocol messages (Q.931, H.245, and registration)
- sending protocol messages (Q.931, H.245, and registration)
- performing T.38 fax server operations

Table 15 shows the valid extension IDs and their purpose.

Table 15. Valid Extension IDs for the gc_Extension( ) Function

Extension ID	Description
IPEXTID_CHANGEMODE	Used with **gc_Extension( )** for the following T.38 fax server operations: • initiating a switch from an audio session to a T.38 fax session • initiating a switch from a T.38 fax session to an audio session • accepting a request to switch from audio to T.38 fax or vice versa • rejecting a request to switch from audio to T.38 fax or vice versa Also used in GCEV_EXTENSION events to provide notification of incoming messages including: • a RequestMode (H.323) or REINVITE (SIP) message indicating a request to switch from audio to T.38 fax • a RequestMode (H.323) or REINVITE (SIP) message indicating a request to switch from T.38 fax to audio • a RequestModeAck (H.323) or 200 OK (SIP) message indicating that a switch to audio or T.38 fax has completed successfully See Section 4.18, "T.38 Fax Server Support", on page 126 for more information.
IPEXTID_FOIP	Used in GCEV_EXTENSION events for notification of information related to fax. See Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105 for more information.
IPEXTID_GETINFO	Used to retrieve call-related information. See Section 4.4, "Retrieving Current Call-Related Information", on page 77 for more information.
IPEXTID_IPPROTOCOL_STATE	Used in GCEV_EXTENSION events for notification of intermediate protocol states, such as, Q.931 and H.245 session connections and disconnections. See Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105 for more information.

Table 15. Valid Extension IDs for the gc_Extension( ) Function

Extension ID	Description
IPEXTID_MEDIAINFO	Used in GCEV_EXTENSION events for notification of the initiation and termination of media streaming in the transmit and receive directions. In the case of media streaming connection notification, the datatype of the parameter is IP_CAPABILITY and consists of the coder configuration that resulted from the capability exchange with the remote peer. See Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105 for more information.
IPEXTID_RECEIVE_DTMF	Used to select which DTMF types, when detected, provide notification to the application. See Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105 for more information.
IPEXTID_RECEIVEMSG	Used in GCEV_EXTENSION events when Q.931, H.245, and non-standard registration messages are received.
IPEXTID_SEND_DTMF	Used to send DTMF digits. When this call is successful, the sending side receives a GCEV_EXTENSIONCMPLT event with the same ext_id. The remote side receives a GCEV_EXTENSION event with IPEXTID_RECEIV_DTMF but only when configured for notification of a specific type of DTMF. See Section 4.11, "Enabling and Disabling Unsolicited Notification Events", on page 105 for more information.
IPEXTID_SENDMSG	Used to send Q.931, H.245, and RAS messages. The supported parameter sets are: • IPSET_MSG_H245 • IPSET_MSG_Q931 • IPSET_MSG_RAS When the **gc_Extension( )** function completes successfully, the sending side receives a GCEV_EXTENSIONCMPLT event with the same ext_id. The remote side receives a GCEV_EXTENSION event with an ext_id field value of IPEXTID_RECEIVEMSG.

The **gc_Extension( )** function is only used in the context of a call where the protocol is already known, therefore the protocol does not need to be specified. When protocol-specific information is specified and it is not of the correct protocol type, for example, attempting to send a Q.931 FACILITY message in a SIP call, the operation fails.

See the Section 4.4.2, "Examples of Retrieving Call-Related Information", on page 81 for a code example showing how to identify the type of extension event and extract the related information.

7.2.8 gc_GetAlarmParm( ) Variances for IP

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 4.15.2, "Retrieving the Media Device Handle", on page 109 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_INFO structure, which includes an IPM_QOS_THRESHOLD_DATA structure that contains the parameters representing threshold values. See the IPM_QOS_THRESHOLD_INFO structure in the *IP Media Library API Library Reference* and the *IP Media Library API Programming Guide* for more information. The thresholds supported by Global Call for HMP are QOSTYPE_LOSTPACKETS, QOSTYPE_JITTER, QOSTYPE_RTCPTIMEOUT, and QOSTYPE_RTPTIMEOUT.

mode
Must be set to EV_SYNC.

Note: Applications **must** include the *gcipmlib.h* header file before Global Call can be used to set or retrieve QoS threshold values.

See Section 4.15.3, "Setting QoS Threshold Values", on page 109 for code examples.

7.2.9 gc_GetCallInfo( ) Variances for IP

The **gc_GetCallInfo( )** function can be used to retrieve calling (ANI) or called party (DNIS) information such as an IP address, an e-mail address, an E.164 number, a URL, or the call identifier (Call ID) used by the underlying protocol to globally, uniquely identify the call. The values of the **info_id** parameter that are supported for both SIP and H.323 are:

ORIGINATION_ADDRESS
the calling party information (equivalent to ANI)

DESTINATION_ADDRESS
the called party information (equivalent to DNIS)

IP_CALLID
the globally unique identifier used by the underlying protocol to identify the call (Call ID or GUID)

Two additional, SIP-specific values for the **info_id** parameter that allow retrieval of information from the From URI and To URI SIP message fields are described below under the "Variance for SIP" heading.

When an **info_id** of ORIGINATION_ADDRESS (ANI) is specified and the function completes successfully, the **valuep** string is a concatenation of values delimited by a pre-determined character (configurable in the IPCCLIB_START_DATA data structure used by **gc_Start( )**; the default is a comma).

When an **info_id** of DESTINATION_ADDRESS (DNIS) is specified and the function completes successfully, the **valuep** string is a concatenation of values delimited by a pre-determined character (configurable in the IPCCLIB_START_DATA data structure used by **gc_Start( )**; the default is a comma). The IP address of the destination gateway (that is processing the DNIS) is **not** included in the string.

When an **info_id** of IP_CALLID (Call ID) is specified and the function completes successfully, the buffer pointed to by the **valuep** argument contains the globally unique identifier used by the underlying protocol to identify the call. The size and datatype of the Call ID depends on the protocol. To assure adequate buffer size when the protocol is unknown, use the IP_CALLIDSIZE define to allocate a buffer that is large enough to hold any type of Call ID value (i.e., either an H.323 array of octets or a SIP string).

Note: For outbound calls the **gc_GetCallInfo( )** function can be used to retrieve valid Call ID information only after the Proceeding state.

The **gc_GetCallInfo( )** function can also be used to query the protocol used by a call. The **info_id** parameter should be set to CALLPROTOCOL and the **valuep** parameter returns a pointer to an integer that is one of the following values:

- CALLPROTOCOL_H323
- CALLPROTOCOL_SIP

Note: For an inbound call, the **gc_GetCallInfo( )** function can be used to determine the protocol any time after the GCEV_OFFERED event is received and before the GCEV_DISCONNECTED event is received.

Variance for H.323

When retrieving calling (ANI) information, the following rules apply. Any section in the string that includes a prefix (TA:, TEL:, or NAME:) has been inserted as an alias by the originating party. Any section in the string that does not include a prefix has been inserted as a **calling party** number (Q.931) by the originating party.

When retrieving called party (DNIS) information, the following rules apply. Any section in the string that includes a prefix (TA:, TEL:, or NAME:) has been inserted as an alias by the originating party. Any section in the string that does not include a prefix has been inserted as a **called party** number (Q.931) by the originating party.

When retrieving Call ID information, the buffer pointed to by the **valuep** argument contains an array of octets. The size of this array is IP_H323_CALLIDSIZE bytes. To assure adequate buffer size when the protocol is unknown, use the IP_CALLIDSIZE define to create a buffer that is large enough to hold any type of Call ID value (i.e., either H.323 or SIP).

Variance for SIP

When retrieving calling part or called party information, prefixes (TA:, TEL; or NAME:) are **not** used.

When retrieving calling party (ANI) information, the address is taken from the SIP From: header, and is accessible in one of two forms by using one of the following parameter IDs in the function call:

ORIGINATION_ADDRESS
: Returns the simple origination address in the form
alice@192.168.1.10

ORIGINATION_ADDRESS_SIP
: Returns a SIP-specific origination address that includes additional From URI parameters and tags. The format used is
sip: alice@192.168.1.10;tag=0-13c4-4059c361-23d07406-72fe

When retrieving called party (DNIS) information, the address is taken from the SIP To: header, and is accessible in one of two forms by using one of the following parameter IDs in the function call:

DESTINATION_ADDRESS
: Returns the simple destination address in the form
user@127.0.0.1

DESTINATION_ADDRESS_SIP
: Returns a SIP-specific destination address that includes additional To URI parameters in the form
sip: userB@127.0.0.1;user=Steve

When retrieving Call ID information, the buffer pointed to by the **valuep** argument contains a NULL-terminated string. The maximum size of this string is IP_SIP_CALLIDSIZE bytes. To assure adequate buffer size when the protocol is unknown, use the IP_CALLIDSIZE define. This will assure the buffer is large enough to hold any type of Call ID value (i.e., either H.323 or SIP).

Retrieving SIP Call ID via gc_GetCallInfo( )

The following code example illustrates retrieval of the SIP Call ID using a **gc_GetCallInfo( )** call.

```
/*
 * Assume the following has been done:
 * 1. device has been opened (e.g. :N_iptB1T1:P_SIP, :N_iptB1T2:P_SIP, etc...)
 * 2. gc_WaitCall() has been issued to wait for a call.
 * 3. gc_GetMetaEvent() or gc_GetMetaEventEx() (Windows) has been called
 *    to convert the event into metaevent.
 * 4. a GCEV_OFFERED has been detected.
 */

#include <stdio.h>
#include <srllib.h>
#include <gclib.h>
#include <gcerr.h>
#include <gcip.h>

/*
 * Assume the 'crn' parameter holds the CRN associated with the detected GCEV_OFFERED event.
 */

int print_call_info(CRN crn)
{
    GC_INFO gc_error_info;             /* GlobalCall error information data */
    char cid_buff[IP_SIP_CALLIDSIZE];  /* buffer large enough to hold SIP Call-ID value */
```

```
    if(gc_GetCallInfo(crn, IP_CALLID, cid_buff) != GC_SUCCESS)
    {
        /* process error return as shown */
        gc_ErrorInfo( &gc_error_info );
        printf ("Error: gc_GetCallInfo(IP_CALLID) on crn: 0x%lx, GC ErrorValue: 0x%hx - %s,"\
                " CCLibID: %i - %s, CC ErrorValue: 0x%lx - %s\n",
                crn, gc_error_info.gcValue, gc_error_info.gcMsg, gc_error_info.ccLibId,
                gc_error_info.ccLibName, gc_error_info.ccValue, gc_error_info.ccMsg);
        return (gc_error_info.gcValue);
    }

    printf ("gc_GetCallInfo(IP_CALLID) on crn: 0x%lx, returned - %s\n", crn, cid_buff);

    return(0);
}
```

7.2.10 gc_GetCTInfo( ) Variances for IP

The **gc_GetCTInfo( )** function can be used to retrieve product information (via the CT_DEVINFO structure) for the media sub-device (ipm) attached to the network device (ipt). If no media device is associated with the network device, the function returns as though not supported.

7.2.11 gc_GetResourceH( ) Variances for IP

The **gc_GetResourceH( )** function can be used to retrieve the media device (ipm 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 *gcipmlib.h* header file before Global Call can be used to set or retrieve QoS threshold values.

The other resource types including GC_NETWORKDEVICE (for a network device), GC_VOICEDEVICE (for a voice device), and GC_NET_GCLINEDEVICE (to retrieve the Global Call line device handle when the media handle is known) are also supported.

Note: The GC_VOICEDEVICE option above applies only if the voice device was opened with the line device or opened separately and subsequently attached to the line device.

7.2.12 gc_GetXmitSlot( ) Variances for IP

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

linedev
: 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 and a media device attached).

sctsinfop
: A pointer to the transmit time slot information for the IP Media device (a pointer to a CT Bus time slot information structure).

7.2.13 gc_InitXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

The **parmblkp** and **ret_rerouting_infopp** parameters are ignored and should be set to NULL. The **gc_InitXfer( )** function returns -1 if invalid parameter are specified. The **gc_InitXfer( )** has an associated GCEV_INIT_XFER termination event that is received on the specified CRN indicating that the initiate transfer request was successful.

7.2.14 gc_InvokeXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

The GCEV_XFER_CMPLT event does not necessarily mean that the transferred call between party B and party C was connected, only that it was confirmed to be delivered. Specifically, it indicates that ALERTING or CONNECT was received from party C on the transferred call.

The parameter variances listed in Table 16 should be noted.

Table 16. gc_InvokeXfer( ) Supported Parameters for H.450.2

Parameter	Meaning
extracrn	For supervised call transfer parameter value must be the CRN of the secondary/consultation call with party C. For blind call transfer parameter value must be zero.
numberstr	Ignored in supervised call transfer – set to NULL. For blind call transfer, used to provide address of party C. Signaled to party B in the GCEV_REQ_XFER event. Format can be: • transport address, for example, “TA:146.152.0.1” • E.164 alias, for example, “TEL:9739674700” • host address, for example, “NAME: myhostname” **Note:** Only a single transport address should be specified in either the char* **numberstr** parameter or the GC_MAKECALL_BLK *makecallp field.

Table 16. gc_InvokeXfer( ) Supported Parameters for H.450.2 (Continued)

Parameter	Meaning
makecallp	Ignored in supervised call transfer – set to NULL. For blind call transfer, used to provide address of party C. Signaled to party B in the GCEV_REQ_XFER event. **Note:** Only a single transport address should be specified in either the char ***numberstr** parameter or the GC_MAKECALL_BLK *makecallp field.
timeout	Ignored. H.450.2 timers (T1, T2, T3, T4) are implicitly maintained at 20 seconds) – set to NULL.

Table 17 to Table 20 list the possible event failure cause values.

Table 17. H.450.2 CtInitiate Errors Received from the Network

ctInitiate Error	Result Values	GC Event
notAvailable	CC: IPEC_H450NotAvailable GC: GCRV_REMOTEREJ_UNAVAIL	GCEV_INVOKE_XFER_REJ
invalidCallState	CC: IPEC_H450InvalidCallState GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_FAIL
invalidReroutingNumber	CC: IPEC_H4502InvalidReroutingNumber GC: GCRV_REMOTEREJ_INVADDR	GCEV_INVOKE_XFER_REJ
unrecognizedCallIdentity	CC: IPEC_H4502UnrecognizedCallIdentity GC: GCRV_REMOTEREJ_INVADDR	GCEV_INVOKE_XFER_FAIL
establishmentFailure	CC: IPEC_H4502EstablishmentFailure GC: GCRV_REMOTEREJ_UNSPECIFIED	GCEV_INVOKE_XFER_FAIL
supplementaryServiceInteractionNotAllowed	CC: IPEC_H450SuppServInteractionNotAllowed GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_REJ
unspecified	CC: IPEC_H4502Unspecified GC: GCRV_REMOTEREJ_UNSPECIFIED	GCEV_INVOKE_XFER_REJ

Table 18. H.450.2 CtIdentify Errors Received From the Network

ctIdentify Error	Result Values	GC Event
notAvailable	CC: IPEC_H450TRTSENotAvailable GC: GCRV_REMOTEREJ_UNAVAIL	GCEV_INVOKE_XFER_REJ
invalidCallState	CC: IPEC_H450TRTSEInvalidCallState GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_FAIL
supplementaryService InteractionNotAllowed	CC: IPEC_H450TRTSESuppServInteractionNotAllowed GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_REJ
unspecified	CC: IPECH4502TRTSEUnspecified GC: GCRV_REMOTEREJ_UNSPECIFIED	GCEV_INVOKE_XFER_REJ

Table 19. H.450.2 CtSetup Errors Received From the Network

ctSetup Error	Result Values	GC Event
notAvailable	CC: IPEC_H450NotAvailable GC: GCRV_REMOTEREJ_UNAVAIL	GCEV_INVOKE_XFER_REJ
invalidCallState	CC: IPEC_H450InvalidCallState GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_FAIL
invalidReroutingNumber	CC: IPEC_H4502InvalidReroutingNumber GC: GCRV_REMOTEREJ_INVADDR	GCEV_INVOKE_XFER_REJ
unrecognizedCallIdentity	CC: IPEC_H4502UnrecognizedCallIdentity GC: GCRV_REMOTEREJ_INVADDR	GCEV_INVOKE_XFER_FAIL
supplementaryServiceInteractionNotAllowed	CC: IPEC_H450SuppServInteractionNotAllowed GC: GCRV_REMOTEREJ_NOTALLOWED	GCEV_INVOKE_XFER_REJ
unspecified	CC: IPEC_H4502Unspecified GC: GCRV_REMOTEREJ_UNSPECIFIED	GCEV_INVOKE_XFER_REJ

Table 20. H.450.2 CT Timer Expiry

Endpoint – Timer	Result Values	GC Event
TRGSE – T1	CC: IPEC_H450CTT1Timeout GC: GCRV_TIMEOUT	GCEV_INVOKE_XFER_FAIL
TRGSE – T3	CC: IPEC_H450CTT3Timeout GC: GCRV_TIMEOUT	GCEV_INVOKE_XFER_FAIL

7.2.15 gc_Listen( ) Variances for IP

The **gc_Listen( )** function is supported in both asynchronous and synchronous modes. The function is blocking in synchronous mode.

Note: For line devices that comprise media (ipm) and voice (dxxx) devices, routing is only done on the media devices. Routing of the voice devices must be done using the Voice API (dx_ functions).

7.2.16 gc_MakeCall( ) Variances for IP

Global Call supports multiple IP protocols on a single IPT Network device. See Section 2.3.3, "IPT Network Devices", on page 36 for more information. When using a multi-protocol network device (that is, one opened in P_IP mode), the application specifies the protocol in the associated GC_MAKECALL_BLK structure, using the set ID IPSET_PROTOCOL, the parameter ID IPPARM_PROTOCOL_BITMASK, and one of the following values:

- IP_PROTOCOL_SIP
- IP_PROTOCOL_H323

A network device that is opened in multi-protocol mode defaults to IP_PROTOCOL_H323 if the protocol is not explicitly set in the makecall block.

Note: Applications should **not** use the **gc_SetUserInfo( )** function to set the IP protocol.

When making calls on devices that support only one protocol, it is not necessary to include an IPSET_PROTOCOL element in the makecall block. If the application tries to include an IPSET_PROTOCOL element in the makecall block that conflicts with the protocol supported by the device, the application receives an error.

When using SIP, if the remote site does not respond to an outgoing INVITE sent by the call control library, the **gc_MakeCall( )** function times out after 32 seconds and generates a GCEV_DISCONNECTED event. In this no-response scenario, if the application attempts to drop the call before the 32 second timeout is reached, the library sends a CANCEL to the remote site. If there is no response by the remote site to the CANCEL, there will be an additional 32 second timeout, at the end of which a GCEV_DISCONNECTED event will be reported.

7.2.16.1 Configurable Call Parameters

Call parameters 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 ext_datap field within the GCLIB_MAKECALL_BLK structure points to a GC_PARM_BLK structure with a list of the parameters to be set as call values. The parameters that can be specified through the ext_datap pointer depend on the protocol used, H.323 or SIP and are described in the subsections following.

Variance for H.323

Table 21 shows the call parameters that can be specified when using **gc_MakeCall( )** with H.323.

Table 21. Configurable Call Parameters When Using H.323

Set ID	Parameter ID(s) and Datatypes
GCSET_CHAN_CAPABILITY	IPPARM_LOCAL_CAPABILITY Datatype IP_CAPABILITY. See the reference page for IP_CAPABILITY on page 235 for more information. **Note:** If no transmit/receive coder type is specified, any supported coder type is accepted.
IPSET_CALLINFO See Section 8.2.2, “IPSET_CALLINFO”, on page 217 for more information.	IPPARM_CONNECTIONMETHOD Enumeration, with one of the following values: • IPPARM_CONNECTIONMETHOD_FASTSTART • IPPARM_CONNECTIONMETHOD_SLOWSTART See Section 4.2, “Using Fast Start and Slow Start Setup”, on page 66 for more information.

Notes:
The term “String” implies the normal definition of a character string which can contain letters, numbers, white space, and a null (for termination).

Table 21. Configurable Call Parameters When Using H.323

Set ID	Parameter ID(s) and Datatypes
	IPPARM_CALLID Array of octets, length = MAX_IP_H323_CALLID_LENGTH
	IPPARM_DISPLAY String, max. length = MAX_DISPLAY_LENGTH (82), null-terminated
	IPPARM_H245TUNNELING Enumeration, with one of the following values: • IP_H245TUNNELING_ON or IP_H245TUNNELING_OFF See Section 4.13, “Enabling and Disabling Tunneling in H.323”, on page 107 for more information.
	IPPARM_PHONELIST String, max. length = 131.
	IPPARM_USERUSER_INFO String, max. length = MAX_USERUSER_INFO_LENGTH (131 bytes)
IPSET_CONFERENCE	IPPARM_CONFERENCE_GOAL Enumeration with one of the following values: • IP_CONFERENCEGOAL_UNDEFINED • IP_CONFERENCEGOAL_CREATE • IP_CONFERENCEGOAL_JOIN • IP_CONFERENCEGOAL_INVITE • IP_CONFERENCEGOAL_CAP_NEGOTIATION • IP_CONFERENCEGOAL_SUPPLEMENTARY_SRVC
Notes: The term “String” implies the normal definition of a character string which can contain letters, numbers, white space, and a null (for termination).	

Table 21. Configurable Call Parameters When Using H.323

Set ID	Parameter ID(s) and Datatypes
IPSET_NONSTANDARDDATA See Section 8.2.16, "IPSET_NONSTANDARDDATA", on page 226 for more information.	Either: • IPPARM_NONSTANDARDDATA_DATA String, max. length = MAX_NS_PARM_DATA_LENGTH (128) and • IPPARM_NONSTANDARDDATA_OBJID Unsigned Int[ ], max. length =MAX_NS_PARM_OBJID_LENGTH (40) or • IPPARM_NONSTANDARDDATA_DATA String, max. length = MAX_NS_PARM_DATA_LENGTH (128) and • IPPARM_H221NONSTANDARD Datatype IP_H221NONSTANDARD
IPSET_NONSTANDARDCONTROL See Section 8.2.15, "IPSET_NONSTANDARDCONTROL", on page 225 for more information.	Either: • IPPARM_NONSTANDARDDATA_DATA String, max. length = MAX_NS_PARM_DATA_LENGTH (128) and • IPPARM_NONSTANDARDDATA_OBJID Unsigned Int[ ], max. length = MAX_NS_PARM_OBJID_LENGTH (40) or • IPPARM_NONSTANDARDDATA_DATA String, max. length = MAX_NS_PARM_DATA_LENGTH (128) and • IPPARM_H221NONSTANDARD Datatype IP_H221NONSTANDARD

Notes:
The term "String" implies the normal definition of a character string which can contain letters, numbers, white space, and a null (for termination).

Variance for SIP

Table 22 shows the call parameters that can be specified when using **gc_MakeCall( )** with SIP.

Table 22. Configurable Call Parameters When Using SIP

Set ID	Parameter ID and Datatype
GCSET_CHAN_CAPABILITY	IPPARM_LOCAL_CAPABILITY Datatype IP_CAPABILITY. See reference page for IP_CAPABILITY on page 235 for more information. **Note:** If no transmit/receive coder type is specified, any supported coder type is accepted.
IPSET_CALLINFO See Section 8.2.2, "IPSET_CALLINFO", on page 217 for more information.	IPPARM_CONNECTIONMETHOD Enumeration, with one of the following values: • IPPARM_CONNECTIONMETHOD_FASTSTART • IPPARM_CONNECTIONMETHOD_SLOWSTART See Section 4.2, "Using Fast Start and Slow Start Setup", on page 66 for more information.
	IPPARM_CALLID String, max. length = MAX_IP_SIP_CALLID_LENGTH **Note:** Directly manipulating the SIP Call ID message header via IPSET_SIP_MSGINFO and IPPARM_CALLID_HDR will override any value provided here.
	IPPARM_DISPLAY String, max. length = MAX_DISPLAY_LENGTH (82), null-terminated
	IPPARM_PHONELIST String, max. length = 131
Notes: The term "String" implies the normal definition of a character string which can contain letters, numbers, white space, and a null (for termination). The parameter names used are more closely aligned with H.323 terminology. Corresponding SIP terminology is described in http://www.ietf.org/rfc/rfc3261.txt?number=3261.	

7.2.16.2 Origination Address Information

The origination address can be specified in the origination field of type GCLIB_ADDRESS_BLK in the GCLIB_MAKECALL_BLK structure. The address field in the GCLIB_ADDRESS_BLK contains the actual address and the address_type field in the GCLIB_ADDRESS_BLK structure defines the type (IP address, name, telephone number) in the address field.

Note: The total length of the address string is limited by the value MAX_ADDRESS_LEN (defined in *gclib.h*).

The origination address can be set using the **gc_SetCallingNum( )** function, which is a deprecated function. The preferred equivalent is **gc_SetConfigData( )**. See the *Global Call API Library Reference* for more information.

7.2.16.3 Forming a Destination Address String

Variance for H.323

The destination address is formed by concatenating values from three different sources:

- the GC_MAKECALL_BLK
- the **numberstr** parameter of **gc_MakeCall( )**
- the phone list

The order or precedence of these elements and the rules for forming a destination address are described below.

Notes: ***1.*** The following description refers to a delimited string. The delimiter is configurable by setting the value of the delimiter field in the IP_CCLIB_START_DATA structure used by the **gc_Start( )** function.

2. The total length of the address string is limited by the value MAX_ADDRESS_LEN (defined in *gclib.h*).

3. The destination address must be a valid address that can be translated by the remote node.

The destination information string is delimited concatenation of the following strings in the order of precedence shown:

1. A string constructed from the destination field of type GCLIB_ADDRESS_BLK in the GCLIB_MAKECALL_BLK. When specifying the destination information in the GCLIB_ADDRESS_BLK, the address field contains the actual address information and the address_type field defines the type (IP address, name, telephone number) in the address. For example, if the address field is “127.0.0.1”, the address_type field must be GCADDRTYPE_IP. The 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 – e-mail address
2. The **numberstr** parameter in the **gc_MakeCall( )** function. The **numberstr** parameter is treated as a free string that may be a delimited concatenation of more than one section. The application may include a prefix in a section that maps to a corresponding field in the Setup message. See Section 7.2.16.4, “Destination Address Interpretation”, on page 179, for more information.
3. Phone list as described in Table 21, “Configurable Call Parameters When Using H.323”, on page 173 (and set using IPSET_CALLINFO, IPPARM_PHONELIST). Phone List is treated as a free string that may be a delimited concatenation of more than one section. The application may prefix a section that maps to a corresponding field in the Setup message. See the Section 7.2.16.4, “Destination Address Interpretation”, on page 179 for more information.

Variance for SIP

The format of the destination address for a SIP call is:

```
user@host; param=value
```

with the elements representing:

user
: A user name or phone number

host
: A domain name or an IP address

param=value
: An optional additional parameter

When making a SIP call, the destination address is formed according to the following rules in the order of precedence shown:

1. If Phone List (as described in Table 22, "Configurable Call Parameters When Using SIP", on page 176 and identified by IPSET_CALLINFO, IPPARM_PHONELIST) exists, it is taken to construct the global destination-address-string.
2. If the destination address field (of type GCLIB_ADDRESS_BLK in GCLIB_MAKECALL_BLK) exists, it is taken to construct the global destination-address-string. The address_type in GCLIB_ADDRESS_BLK is ignored. If the global destination-address-string is not empty before setting the parameter, an "@" delimiter is used to separate the two parts.
3. If the **numberstr** parameter from the **gc_MakeCall( )** function exists, it is taken to destination-address-string. If the global destination-address-string is not empty before setting the parameter, a ";" delimiter is used to separate the two parts.

Note: To observe the logic described above, the application may use only one of the APIs to send a string that is a valid SIP address.

The following code examples demonstrate the recommended ways of forming the destination string when making a SIP call. Prerequisite code for setting up the GC_MAKECALL_BLK in all the scenarios described in this section is as follows:

```
GC_MAKECALL_BLK gcmkbl;
GCLIB_MAKECALL_BLK gclib_mkbl = {0};
gcmkbl.cclib = NULL;
gcmkbl.gclib = &gclib_mkbl;
GC_PARM_BLK *target_datap = NULL;

gc_util_insert_parm_val(&target_datap,
                        IPSET_PROTOCOL,
                        IPPARM_PROTOCOL_BITMASK,
                        sizeof(char),
                        IP_PROTOCOL_SIP);
```

Scenario 1 – Making a SIP call to a known IP address, where the complete address (user@host) is specified in the makecall block:

```
char *pDestAddrBlk = "11223344@127.0.0.1";  /* where "11223344" is the
                                               phone number of the user
                                               and "127.0.0.1" is the
                                               IP address of the host */
```

```
/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_TRANSPARENT;

/* calling the function with the MAKECALL_BLK, and numberstr parameter=NULL
   the INVITE dest address will be: 11223344@127.0.0.1 */
gc_MakeCall(ldev, &crn, NULL, &gcmkbl, MakeCallTimeout, EV_ASYNC);
```

Scenario 2 – Making a SIP call to a known IP address, where the complete address (user@host) is formed by the combination of the destination address in the makecall block and the phone list element:

```
char *pDestAddrBlk = "127.0.0.1";   /*host*/
char *IpPhoneList = "003227124311"; /*user*/

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

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_TRANSPARENT;

gclib_mkbl.ext_datap = target_datap;

/* calling the function with the MAKECALL_BLK, and numberstr parameter = NULL
   the INVITE dest address will be: 003227124311@127.0.0.1 */
gc_MakeCall(ldev, &crn, NULL, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 3 – Making a SIP call to a known IP address, where the complete address (user@host) is formed by the combination of the destination address in the makecall block, a phone list element, and optional parameter (user=phone):

```
char *pDestAddrBlk = "127.0.0.1";  /*host*/
char *IpPhoneList= "003227124311"; /*user*/
char *pDestAddrStr = "user=phone"; /*extra parameter*/

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

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_TRANSPARENT;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK, and numberstr parameter = NULL
   the INVITE dest address will be: 003227124311@127.0.0.1;user=phone */
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

7.2.16.4 Destination Address Interpretation

Note: The following information applies when using H.323 only.

Once a destination string is formed as described in the previous section, the H.323 stack treats the string according to the following rules:

- The **first** section of the string is the destination of the next IP entity (for example, a gateway, terminal, the alias for a remote registered entity, etc.) with which the application attempts to negotiate.
- A non-prefixed section in the string is the Q.931 calledPartyNumber and is the **last** section that is processed. Any section following the first non-prefixed section is ignored. Only **one** Q.931 calledPartyNumber is allowed in the destination string.
- One or more prefixed sections (H.225 destinationAddress fields) must appear **before** the non-prefixed section (Q.931 calledPartyNumber).
- When using free strings (**numberstr** parameter or Phone List), if the application wants to prefix buffers, valid buffer prefixes for H.225 addresses are:
  - TA: – IP transport address
  - TEL: – e164 telephone number
  - NAME: – H.323 ID
  - URL: – Universal Resource Locator
  - EMAIL: – e-mail address

The following code examples demonstrate the recommended ways of forming the destination string when making an H.323 call. Prerequisite code for setting up the GC_MAKECALL_BLK in all the scenarios described in this section is as follows:

```
GC_MAKECALL_BLK gcmkbl;
GCLIB_MAKECALL_BLK gclib_mkbl = {0};
gcmkbl.cclib = NULL;
gcmkbl.gclib = &gclib_mkbl;
GC_PARM_BLK *target_datap = NULL;

gc_util_insert_parm_val(&target_datap,
                        IPSET_PROTOCOL,
                        IPPARM_PROTOCOL_BITMASK,
                        sizeof(char),
                        IP_PROTOCOL_H323);
```

Scenario 1 – Making a call to a known IP address, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = "127.0.0.1";
char *pDestAddrStr = "123456";

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_IP;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK*/
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 2 – Making a call to a known IP address, setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = "127.0.0.1";
char *pDestAddrStr = "TEL:111,TEL:222,76543";

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_IP;
```

```
gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK*/
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 3 – Making a call to a known IP address, setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = "127.0.0.1";
char *pDestAddrStr = "TEL:111,TEL:222,NAME:myName";
char *IpPhoneList= "003227124311";

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

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_IP;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK*/
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 4 – Making a call to a known IP address, setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = "127.0.0.1";
char *IpPhoneList= "TEL:003227124311,TEL:444,TEL:222,TEL:1234,171717";
/* insert phone list */
gc_util_insert_parm_ref(&target_datap,
                        IPSET_CALLINFO,
                        IPPARM_PHONELIST,
                        (unsigned char)(strlen(IpPhoneList)+1),
                        IpPhoneList);
gclib_mkbl.ext_datap = target_datap;

/* set GCLIB_ADDRESS_BLK with destination string & type*/
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_IP;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK, and numberstr
   parameter = NULL */
gc_MakeCall(ldev, &crn, NULL, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 5 – While registered, making a call, via the gatekeeper, to a registered entity (using a known H.323 ID), setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = " RegisteredRemoteGW ";  /* The alias of the remote (registered) entity */
char *pDestAddrStr = "TEL:111,TEL:222,987654321";

/* set GCLIB_ADDRESS_BLK with destination string & type (H323-ID) */
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_DOMAIN;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK */
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 6 – While registered, making a call, via the gatekeeper, to a registered entity (using a known e-mail address), setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = " user@host.com ";  /* The alias of the remote (registered) entity */
char *pDestAddrStr = "TEL:111,TEL:222,987654321";

/* set GCLIB_ADDRESS_BLK with destination string & type (EMAIL) */
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_EMAIL;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK */
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

Scenario 7 – While registered, making a call, via the gatekeeper, to a registered entity (using a known URL), setting a number of H.225 aliases, and setting the Q.931 calledPartyNumber:

```
char *pDestAddrBlk = "www.gw1.intel.com";  /* The alias of the remote (registered) entity */
char *pDestAddrStr = "TEL:111,TEL:222,987654321";

/* set GCLIB_ADDRESS_BLK with destination string & type (URL) */
strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
gcmkbl.gclib->destination.address_type = GCADDRTYPE_URL;

gclib_mkbl.ext_datap = target_datap;
/* calling the function with the MAKECALL_BLK */
gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl, MakeCallTimeout,EV_ASYNC);
```

7.2.16.5 Specifying a Timeout

Note: The following information applies when using H.323 only.

The **timeout** parameter of the **gc_MakeCall( )** function specifies the maximum time in seconds to wait for the establishment of a new call, after receiving the first response to the call. This value corresponds to the **Q.931\connectTimeOut** parameter. If the call is not established during this time, the Disconnect procedure is initiated. The default value is 120 seconds.

In addition to the **Q.931\connectTimeOut** parameter described in Section 7.2.16, "gc_MakeCall( ) Variances for IP", on page 172, two other parameters that affect the timeout behavior, but are not configurable are:

Q931\responseTimeOut
: The 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 4 seconds.

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 seconds.

Note: When using the H.323 protocol, the application may receive a timeout when trying to make an outbound call if network congestion is encountered and a TCP connection cannot be established. In this case, the SETUP message is not sent on the network.

7.2.16.6 Code Examples

H.323-Specific Code Example

The following code example shows how to make a call using the H.323 protocol.

```
/* Make an H323 IP call on line device ldev */
void MakeH323IpCall(LINEDEV ldev)
{
   char *IpDisplay = "This is a Display"; /* display data */
   char *IpPhoneList= "003227124311"; /* phone list */
   char *IpUUI = "This is a UUI";    /* user to user information string */
   char *pDestAddrBlk = "127.0.0.1"; /* destination IP address for MAKECALL_BLK*/
   char *pSrcAddrBlk = "987654321";  /* origination address for MAKECALL_BLK*/
   char *pDestAddrStr = "123456";    /* destination string for gc_MakeCall() function*/
   char *IpNSDataData = "This is an NSData data string";
   char *IpNSControlData = "This is an NSControl data string";
   char *IpCommonObjId = "1 22 333 4444"; /* unique format */
   IP_H221NONSTANDARD appH221NonStd;
   appH221NonStd.country_code = 181; /* USA */
   appH221NonStd.extension = 11;
   appH221NonStd.manufacturer_code = 11;
   int ChoiceOfNSData = 1;
   int ChoiceOfNSControl = 1;
   int rc = 0;
   CRN crn;
   GC_MAKECALL_BLK gcmkbl;
   int MakeCallTimeout = 120;

   /* 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;

   /* set GCLIB_ADDRESS_BLK with destination string & type*/
   strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
   gcmkbl.gclib->destination.address_type = GCADDRTYPE_IP;

   /* set GCLIB_ADDRESS_BLK with origination string & type*/
   strcpy(gcmkbl.gclib->origination.address,pSrcAddrBlk);
   gcmkbl.gclib->origination.address_type = GCADDRTYPE_NAT;

   /* set signaling PROTOCOL to H323. default is H323 if device is multi-protocol */
   rc = gc_util_insert_parm_val(&target_datap,
                                IPSET_PROTOCOL,
                                IPPARM_PROTOCOL_BITMASK,
                                sizeof(char),
                                IP_PROTOCOL_H323);

   /* 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);

t_Capability.type = GCCAPTYPE_AUDIO;
t_Capability.direction = IP_CAP_DIR_LCLRECEIVE;
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,
                             (unsigned char)(strlen(IpDisplay)+1),
                             IpDisplay);

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

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

    /* setting NS Data elements */
    gc_util_insert_parm_ref(&target_datap,
                           IPSET_NONSTANDARDDATA,
                           IPPARM_NONSTANDARDDATA_DATA,
                           (unsigned char)(strlen(IpNSDataData)+1),
                           IpNSDataData);

    if(ChoiceOfNSData) /* App chooses in advance which type of */
    {                  /* second NS element to use */
       gc_util_insert_parm_ref(&target_datap,
                               IPSET_NONSTANDARDDATA,
                               IPPARM_H221NONSTANDARD,
                               sizeof(IP_H221NONSTANDARD),
                               &appH221NonStd);
    }

    else
    {
       gc_util_insert_parm_ref(&target_datap,
                               IPSET_NONSTANDARDDATA,
                               IPPARM_NONSTANDARDDATA_OBJID,
                               (unsigned char)(strlen(IpCommonObjId)+1),
                               IpCommonObjId);
    }

    /* setting NS Control elements */
    gc_util_insert_parm_ref(&target_datap,
                            IPSET_NONSTANDARDCONTROL,
                            IPPARM_NONSTANDARDDATA_DATA,
                            (unsigned char)(strlen(IpNSControlData)+1),
                            IpNSControlData);

    if(ChoiceOfNSControl) /* App chooses in advance which type of */
    {                     /* second NS element to use */
       gc_util_insert_parm_ref(&target_datap,
                               IPSET_NONSTANDARDCONTROL,
                               IPPARM_H221NONSTANDARD,
                               sizeof(IP_H221NONSTANDARD),
                               &appH221NonStd);
    }
    else
    {
       gc_util_insert_parm_ref(&target_datap,
                               IPSET_NONSTANDARDCONTROL,
                               IPPARM_NONSTANDARDDATA_OBJID,
                               (unsigned char)(strlen(IpCommonObjId)+1),
                               IpCommonObjId);
    }

    if(rc == 0)
    {
       gclib_mkbl.ext_datap = target_datap;
       rc = gc_MakeCall(ldev, &crn, pDestAddrStr, &gcmkbl,
                        MakeCallTimeout, EV_ASYNC);

       /* deallocate GC_PARM_BLK pointer */
       gc_util_delete_parm_blk(target_datap);
    }
}
```

SIP-Specific Code Example

The following code example shows how to make a call using the SIP protocol.

```
/* Make a SIP IP call on line device ldev */
void MakeSipIpCall(LINEDEV ldev)
{
   char *IpDisplay = "This is a Display";  /* display data */
   char *pDestAddrBlk = "12345@127.0.0.1"; /* destination IP address for MAKECALL_BLK */
   char *pSrcAddrBlk = "987654321"; /* origination address for MAKECALL_BLK*/

   int rc = 0;
   CRN crn;
   GC_MAKECALL_BLK gcmkbl;
   int MakeCallTimeout = 120;

   /* 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;

   /* set GCLIB_ADDRESS_BLK with destination string & type*/
   strcpy(gcmkbl.gclib->destination.address,pDestAddrBlk);
   gcmkbl.gclib->destination.address_type = GCADDRTYPE_TRANSPARENT;

   /* set GCLIB_ADDRESS_BLK with origination string & type*/
   strcpy(gcmkbl.gclib->origination.address,pSrcAddrBlk);
   gcmkbl.gclib->origination.address_type = GCADDRTYPE_TRANSPARENT;

   /* set signaling PROTOCOL to SIP*/
   rc = gc_util_insert_parm_val(&target_datap,
                                 IPSET_PROTOCOL,
                                 IPPARM_PROTOCOL_BITMASK,
                                 sizeof(char),
                                 IP_PROTOCOL_SIP);

   /* 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);

   t_Capability.type = GCCAPTYPE_AUDIO;
   t_Capability.direction = IP_CAP_DIR_LCLRECEIVE;
   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,
                             (unsigned char)(strlen(IpDisplay)+1),
                             IpDisplay);

if (rc == 0)
{
   gclib_mkbl.ext_datap = target_datap;
   /* numberstr parameter may be NULL if MAKECALL_BLK is set, as secondary
      address is ignored in SIP */
   rc = gc_MakeCall(ldev, &crn, NULL, &gcmkbl, MakeCallTimeout,EV_ASYNC);

   /* deallocate GC_PARM_BLK pointer */
   gc_util_delete_parm_blk(target_datap);
}
}
```

7.2.17 gc_OpenEx( ) Variances for IP

The **gc_OpenEx( )** function is supported in both asynchronous and synchronous mode. Using the function is asynchronous mode is recommended. The procedure for opening devices is the same regardless of whether H.323 or SIP is used. The IPT network device (N_ipt_BxTy) and IP Media device (M_ipmBxCy) can be opened in the same **gc_OpenEx( )** call and a voice device (V_dxxxBwCz) can also be included.

The format of the **devicename** parameter is:

```
:P_nnnn:N_iptBxTy:M_ipmBxCy:V_dxxxBwCz
```

Notes: ***1.*** The board and timeslot numbers for network devices do **not** have to be the same as the board and channel numbers for media devices.

2. It is possible to specify :N_iptBx (without any :M component) in the **devicename** parameter to get an IPT board device handle. Certain Global Call functions, such as **gc_SetConfigData( )**, use the IPT board device to specify call parameters (such as coders) for all devices in one operation or **gc_ReqService( )** to perform registration and deregistration operations. See Section 7.2.24, "gc_SetConfigData( ) Variances for IP", on page 195 and Section 7.2.21, "gc_ReqService( ) Variances for IP", on page 190 for more information.

3. It is also possible to specify :M_ipmBx (without any :N component) in the **devicename** parameter to get an IP Media board device handle.

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_nnnn

Specifies the IP protocol to be used by the device. This field is mandatory. Possible values are:
- P_H323 – Use the device for H.323 calls only
- P_SIP – Use the device for SIP calls only
- P_IP – Multi-protocol option; use the device for SIP or H.323 calls

Note: When specifying an IPT board device (see below), use the multi-protocol option, P_IP.

N_iptBxTy

Specifies the name of the IPT network device where **x** is the logical board number and **y** is the logical channel number. An IPT board device can be specified using N_iptBx, where **x** is the logical board number.

M_ipmBxCy

Specifies the name of the IP Media device, where **x** is the logical board number and **y** is the logical channel number to be associated with an IPT network device. This field is optional.

V_dxxxBwCz

Specifies a voice resource, where **w** and **z** are the voice board and channel numbers respectively. This field is optional.

In other technologies, an IPT board device can be used for alarms. However, for IP technology, the use of an IPT board device (iptBx) for alarms is not supported. When using Global Call with IP, alarms are reported on IP Media (ipm) devices, not IPT network (ipt) devices.

For Windows operating systems, the SRL function **sr_getboardcnt( )** can be used to retrieve the number of IPT board devices in the system. The **class_namep** parameter in this context should be DEV_CLASS_IPT. The SRL function **ATDV_SUBDEVS( )** can be used to retrieve the number of channels on a board. The **dev** parameter in this context should be an IPT board device handle, that is, a handle returned by **gc_OpenEx( )** when opening an IPT board device.

For Linux operating systems, the SRL device mapper functions **SRLGetAllPhysicalBoards( )**, **SRLGetVirtualBoardsOnPhysicalBoard( )** and **SRLGetSubDevicesOnVirtualBoard( )** can be used to retrieve information about the boards and devices in the system.

7.2.18 gc_RejectInitXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

The parameter **parmblkp** is ignored for IP technology and should be set to NULL.

The **gc_RejectInitXfer( )** function can be used at party C only on the receipt of GCEV_REQ_INIT_XFER.

Four of the six Global Call reasons are supported and result in the following CtIdentify error values signaled back to party A. Values GCVAL_REJREASON_INVADDR and GCVAL_REJREASON_INSUFFINFO cause the function to fail with a subsequent error code of IPERR_BAD_PARAM.

Table 23 lists the CtIdentity error codes that are signaled to party A based on the value of the **reason** parameter passed when the **gc_RejectXfer( )** function is called.

Table 23. CtIdentify Errors Signaled From gc_RejectInitXfer( ) to the Network

GC Value	CtIdentify Error
GCVAL_REJREASON_UNSPECIFIED	unspecified
GCVAL_REJREASON_UNAVAIL	notAvailable
GCVAL_REJREASON_INVADDR	NA (will return invalid parameter error)
GCVAL_REJREASON_NOTALLOWED	supplementaryServiceInteractionNotAllowed
GCVAL_REJREASON_INSUFFINFO	NA (will return invalid parameter error)
GCVAL_REJREASON_NOTSUBSCRIBED	supplementaryServiceInteractionNotAllowed

7.2.19 gc_RejectXfer( ) Variances for IP

Note: The information in this section applies when using the H.450.2 protocol (part of H.323) only.

The parameter **parmblkp** is ignored for IP technology.

The **gc_RejectXfer( )** function can be used at party B only on the receipt of GCEV_REQ_XFER.

All six Global Call reasons are supported and result in the following CtInitiate error values signaled back to party A.

Table 24 lists the CtInitiate error codes that are signaled to party A based on the value of the **reason** parameter passed when the **gc_RejectXfer( )** function is called.

Table 24. CtInitiate Errors Signaled From gc_RejectXfer( ) to the Network

GC Value	CtInitiate Error
GCVAL_REJREASON_UNSPECIFIED	unspecified
GCVAL_REJREASON_UNAVAIL	notAvailable
GCVAL_REJREASON_INVADDR	invalidReroutingNumber
GCVAL_REJREASON_NOTALLOWED	supplementaryServiceInteractionNotAllowed
GCVAL_REJREASON_INSUFFINFO	invalidReroutingNumber
GCVAL_REJREASON_NOTSUBSCRIBED	supplementaryServiceInteractionNotAllowed

7.2.20 gc_ReleaseCallEx( ) Variances for IP

The **gc_ReleaseCallEx( )** function is supported in both asynchronous and synchronous mode. Using the function in asynchronous mode is recommended.

Note: An existing call on a line device must be released before an incoming call can be processed.

7.2.21 gc_ReqService( ) Variances for IP

The **gc_ReqService**( ) function can be used to register an endpoint with a registration server (gateway in H.323 or registrar in SIP). Function parameter must be set as follows:

target_type
GCTGT_GCLIB_NETIF

target_ID
An IPT board device, obtained by using **gc_OpenEx( )** with a **devicename** parameter of "N_iptBx"

service_ID
Any valid reference to an unsigned long; must not be NULL

reqdatap
A pointer to a GC_PARM_BLK containing registration information.

respdatapp
Set to NULL for asynchronous mode. This function is not supported in synchronous mode.

mode
EV_ASYNC

The registration information that can be included is protocol specific as described in Table 25 and Table 26, below.

Registration options include:

- Overriding an existing registration value.
  In this case, IPPARM_OPERATION_REGISTER = IP_REG_SET_INFO.

- Adding a registration value.
  In this case, IPPARM_OPERATION_REGISTER = IP_REG_ADD_INFO.
- Removing a registration value; local alias or supported prefix only.
  In this case, IPPARM_OPERATION_REGISTER = IP_REG_DELETE_BY_VALUE.

See Section 4.16.4, “Registration Code Example”, on page 117 for more information.

The **gc_ReqService( )** function also provides the following deregister options:

- Deregister and keep the registration information locally. In this case, IPPARM_OPERATION_DEREGISTER = IP_REG_MAINTAIN_LOCAL_INFO
- Deregister and discard the registration information locally.
  In this case, IPPARM_OPERATION_DEREGISTER = IP_REG_DELETE_ALL

See Section 4.16.5, “Deregistration Code Example”, on page 119 for more information.

Since some of the registration data may be protocol specific, there is a facility to set the protocol type using IP parameters in **reqdatap** and **respdatapp**, which are of type GC_PARM_BLK.

The relevant items for the GC_PARM_BLK are the IPSET_PROTOCOL parameter set ID and the IPPARM_PROTOCOL_BITMASK parameter ID with one of the following values:

- IP_PROTOCOL_H323
- IP_PROTOCOL_SIP
- IP_PROTOCOL_H323 | IP_PROTOCOL_SIP

Note: The default value for the protocol, when not specified by the application, is IP_PROTOCOL_H323.

The GCEV_SERVICERESP event indicates that a service has been responded to by an H.323 gatekeeper or a SIP registrar. The event is received on an IPT board device handle. The event data includes a specification of the protocol used, using the IPSET_PROTOCOL parameter set ID and the IPPARM_PROTOCOL_BITMASK parameter ID with one of the following values:

- IP_PROTOCOL_H323
- IP_PROTOCOL_SIP

Variance for H.323

When using H.323, the registration information that can be included in the GC_PARM_BLK associated with the **gc_ReqService( )** function is shown in Table 25.

Table 25. Registration Information When Using H.323

Set ID	Parameter IDs
GCSET_SERVREQ	PARM_REQTYPE † Datatype IP_REQTYPE_REGISTRATION
GCSET_SERVREQ	PARM_ACK †
† indicates mandatory parameters. These parameters are required to support the generic service request mechanism provided by Global Call and are not sent in any registration message.	

Table 25. Registration Information When Using H.323 (Continued)

Set ID	Parameter IDs
IPSET_PROTOCOL	IPPARM_PROTOCOL_BITMASK Bitmask composed from the following values: • IP_PROTOCOL_H323 • IP_PROTOCOL_SIP
IPSET_REG_INFO See Section 8.2.18, "IPSET_REG_INFO", on page 227, for more information.	IPPARM_OPERATION_REGISTER One of the following values: • IP_REG_SET_INFO • IP_REG_ADD_INFO • IP_REG_DELETE_BY_VALUE IPPARM_OPERATION_DEREGISTER One of the following values: • IP_REG_MAINTAIN_LOCAL_INFO • IP_REG_DELETE_ALL IPPARM_REG_ADDRESS Datatype IP_REGISTER_ADDRESS See the reference page for IP_REGISTER_ADDRESS on page 242 for more information IPPARM_REG_TYPE One of the following values: • IP_REG_GATEWAY • IP_REG_TERMINAL
IPSET_LOCAL_ALIAS	IPPARM_ADDRESS_DOT_NOTATION IPPARM_ADDRESS_EMAIL IPPARM_ADDRESS_H323_ID IPPARM_ADDRESS_PHONE IPPARM_ADDRESS_TRANSPARENT IPPARM_ADDRESS_URL Datatype: String
IPSET_SUPPORTED_PREFIXES	IPPARM_ADDRESS_DOT_NOTATION IPPARM_ADDRESS_EMAIL IPPARM_ADDRESS_H323_ID IPPARM_ADDRESS_PHONE IPPARM_ADDRESS_TRANSPARENT IPPARM_ADDRESS_URL Datatype: String
† indicates mandatory parameters. These parameters are required to support the generic service request mechanism provided by Global Call and are not sent in any registration message.	

Multiple aliases and supported prefix information is supported when the target protocol for registration is H.323.

Variance for SIP

When using SIP, the registration information that can be included in the GC_PARM_BLK associated with the **gc_ReqService( )** function is shown in Table 26.

Table 26. Registration Information When Using SIP

Set ID	Parameter IDs
GCSET_SERVREQ	PARM_REQTYPE † Datatype IP_REQTYPE_REGISTRATION
GCSET_SERVREQ	PARM_ACK †
IPSET_LOCAL_ALIAS	IPPARM_ADDRESS_DOT_NOTATION IPPARM_ADDRESS_EMAIL IPPARM_ADDRESS_H323_ID IPPARM_ADDRESS_PHONE IPPARM_ADDRESS_TRANSPARENT IPPARM_ADDRESS_URL Datatype: String
IPSET_PROTOCOL	IPPARM_PROTOCOL_BITMASK Bitmask composed from the following values: • IP_PROTOCOL_H323 • IP_PROTOCOL_SIP
IPSET_REG_INFO See Section 8.2.18, "IPSET_REG_INFO", on page 227, for more information.	IPPARM_OPERATION_REGISTER One of the following values: • IP_REG_SET_INFO • IP_REG_ADD_INFO • IP_REG_DELETE_BY_VALUE IPPARM_OPERATION_DEREGISTER One of the following values: • IP_REG_MAINTAIN_LOCAL_INFO • IP_REG_DELETE_ALL IPPARM_REG_ADDRESS Datatype IP_REGISTER_ADDRESS See the reference page for IP_REGISTER_ADDRESS on page 242 for more information
† indicates mandatory parameters. These parameters are required to support the generic service request mechanism provided by Global Call and are not sent in any registration message.	

Only one alias is supported when the target protocol for registration is SIP. Prefix information is **not** supported for SIP.

When using SIP, periodic registration is supported. The call control library will automatically re-register every time_to_live/2 seconds.

7.2.22 gc_RespService( ) Variances for IP

The **gc_RespService( )** function operates on an IPT board device and is used to respond to requests from an H.323 gatekeeper or a SIP registrar. Since some of the data may be protocol specific (in future releases), there is a facility to set the protocol type using IP parameters in **datap**, which is of type GC_PARM_BLK.

The following are the relevant function parameters:

target_type
GCTGT_CCLIB_NETIF

target_id
IPT board device

The relevant items for the GC_PARM_BLK are the IPSET_PROTOCOL parameter set ID and the IPPARM_PROTOCOL_BITMASK parameter ID with one of the following values:

- IP_PROTOCOL_H323
- IP_PROTOCOL_SIP
- IP_PROTOCOL_H323 | IP_PROTOCOL_SIP

Note: The default value for the protocol, when not specified by the application, is IP_PROTOCOL_H323.

The GCEV_SERVICEREQ event indicates that a service has been requested by an H.323 gatekeeper or a SIP registrar. The event is received on an IPT board device handle. The event data includes a specification of the protocol used, using the IPSET_PROTOCOL parameter set ID and the IPPARM_PROTOCOL_BITMASK parameter ID with one of the following values:

- IP_PROTOCOL_H323
- IP_PROTOCOL_SIP

7.2.23 gc_SetAlarmParm( ) Variances for IP

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 4.15.2, "Retrieving the Media Device Handle", on page 109 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_INFO structure, which includes an IPM_QOS_THRESHOLD_DATA structure that contains the parameters representing threshold values. See the IPM_QOS_THRESHOLD_INFO structure in the *IP*

Media Library API Library Reference and the *IP Media Library API Programming Guide* for more information. The thresholds supported by Global Call for HMP are QOSTYPE_LOSTPACKETS, QOSTYPE_JITTER, QOSTYPE_RTCPTIMEOUT, and QOSTYPE_RTPTIMEOUT.

mode
: Must be set to EV_SYNC.

Note: Applications **must** include the *gcipmlib.h* header file before Global Call can be used to set or retrieve QoS threshold values.

See Section 4.15.3, “Setting QoS Threshold Values”, on page 109 for code examples.

7.2.24 gc_SetConfigData( ) Variances for IP

The **gc_SetConfigData( )** function is used for a number of different purposes:

- setting parameters for all board devices, including devices that are already open
- enabling and disabling unsolicited GCEV_EXTENSION events on a board device basis
- setting the type of DTMF support and the RFC 2833 payload type on a board device basis
- setting T.38 fax server operating mode
- masking call state events on a line device basis

Notes:

1. The **gc_SetConfigData( )** function operates on board devices, that is, devices opened using **gc_OpenEx( )** with :N_iptBx:P_IP in the **devicename** parameter. By its nature, a board device is multi-protocol, that is, it applies to both the H.323 and SIP protocols and is not directed to one specific protocol. You *cannot* open a board device (with :P_H323 or :P_SIP in the **devicename** parameter) to target a specific protocol.
2. When using the **gc_SetConfigData( )** function to set parameters, the parameter values apply to all board devices, including devices that are already open. The parameters can be overridden by specifying new values in the **gc_SetUserInfo( )** function (on a per line device basis) or the **gc_MakeCall( )** function (on a per call basis).
3. Coder information can be specified for a device when using **gc_SetConfigData( )**, or when using **gc_MakeCall( )** to make a call, or when using **gc_AnswerCall( )** to answer a call.
4. Use **gc_SetUserInfo( )** to set parameters on line devices.

When using the **gc_SetConfigData( )** function on a board device (the first four bullets above), use the following function parameter values:

target_type
: GCTGT_CCLIB_NETIF

target_id
: An IPT board device that can be obtained by using the **gc_OpenEx( )** function with :N_iptBx:P_IP in the **devicename** parameter. See Section 7.2.17, “gc_OpenEx( ) Variances for IP”, on page 187 for more information.

target_datap
: A pointer to a GC_PARM_BLKP structure that contains the parameters to be configured. The parameters that can be included in the GC_PARM_BLK are protocol specific. See the following "Variance for H.323" and "Variance for SIP" sections.

As in other technologies supported by Global Call, the **gc_SetConfigData( )** function can be used to mask call state events, such as GCEV_ALERTING, on a line device basis. When used for this purpose, the **target_type** is GCTGT_GCLIB_CHAN and the **target_ID** is a line device. See the *Call State Event Configuration* section in the *Global Call API Library Reference* for more information on masking events in general.

Variance for H.323

Table 25 describes the call parameters that can be included in the GC_PARM_BLK associated with the **gc_SetConfigData( )** function. These parameters are in addition to the call parameters described in Table 21, "Configurable Call Parameters When Using H.323", on page 173 that can also be included.

Table 27. Parameters Configurable Using gc_SetConfigData( ) When Using H.323

Set ID	Parameter IDs	Use Before†
GCSET_CALL_CONFIG	GCPARM_CALLPROC †† Enumeration with one of the following values: • GCCONTROL_APP – The application must use **gc_CallAck( )** to send the Proceeding message. This is the default. • GCCONTROL_TCCL – The stack sends the Proceeding message automatically.	**gc_AnswerCall( )**
IPSET_CALLINFO	IPPARM_H245TUNNELING ††† Enumeration with one of the following values: • IP_H245TUNNELINGON • IP_H245TUNNELINGOFF	**gc_AnswerCall( )**
IPSET_CONFIG	IPPARM_OPERATING_MODE Enumeration with one of the following values: • IP_AUTOMATIC_MODE • IP_MANUAL_MODE	**gc_AnswerCall( )** **gc_MakeCall( )**
IPSET_DTMF	IPPARM_SUPPORT_DTMF_BITMASK Datatype: Uint8[ ] IPPARM_DTMF_RFC2833_PAYLOAD_TYPE Datatype: Uint8[ ]	**gc_AnswerCall( )** **gc_MakeCall( )**

† Information can be set in any state but it is only used in certain states. See Section 7.1, "Global Call Functions Supported by IP", on page 153 for more information.
†† This is a system configuration parameter for the terminating side, not a call configuration parameter. It cannot be overwritten by setting a new value in **gc_SetUserInfo( )** or **gc_MakeCall( )**.
††† Applies to the configuration of tunneling for inbound calls only. See Section 4.13, "Enabling and Disabling Tunneling in H.323", on page 107 for more information.

Table 27. Parameters Configurable Using gc_SetConfigData( ) When Using H.323 (Continued)

Set ID	Parameter IDs	Use Before†
IPSET_VENDORINFO	IPPARM_VENDOR_PRODUCT_ID String, max. length = MAX_PRODUCT_ID_LENGTH (32) IPPARM_VENDOR_VERSION_ID String, max. length = MAX_VERSION_ID_LENGTH (32) IPPARM_H221NONSTD Datatype IP_H221NONSTANDARD.	**gc_AnswerCall( )** **gc_MakeCall( )**
IPSET_EXTENSIONEVT_MSK	GCACT_ADDMSK Datatype: Uint8[ ] GCACT_SETMSK Datatype: Uint8[ ] GCACT_SUBMSK Datatype: Uint8[ ]	**gc_AnswerCall( )**

† Information can be set in any state but it is only used in certain states. See Section 7.1, “Global Call Functions Supported by IP”, on page 153 for more information.
†† This is a system configuration parameter for the terminating side, not a call configuration parameter. It cannot be overwritten by setting a new value in **gc_SetUserInfo( )** or **gc_MakeCall( )**.
††† Applies to the configuration of tunneling for inbound calls only. See Section 4.13, “Enabling and Disabling Tunneling in H.323”, on page 107 for more information.

Variance for SIP

Table 28 describes the call parameters that can be included in the GC_PARM_BLK associated with the **gc_SetConfigData**( ) function. These parameters are in addition to the call parameters described in Table 22, “Configurable Call Parameters When Using SIP”, on page 176 that can also be included.

Table 28. Parameters Configurable Using gc_SetConfigData( ) When Using SIP

Set ID	Parameter IDs	Use Before†
GCSET_CALL_CONFIG	GCPARM_CALLPROC †† Enumeration with one of the following values: • GCCONTROL_APP – The application must use **gc_CallAck( )** to send the Proceeding message. This is the default. • GCCONTROL_TCCL – The stack sends the Proceeding message automatically.	**gc_AnswerCall( )**
IPSET_CONFIG	IPPARM_OPERATING_MODE Enumeration with one of the following values: • IP_AUTOMATIC_MODE • IP_MANUAL_MODE	**gc_AnswerCall( )** **gc_MakeCall( )**

† Information can be set in any state but it is only used in certain states. See Section 7.1, “Global Call Functions Supported by IP”, on page 153 for more information.
†† This is a system configuration parameter for the terminating side, not a call configuration parameter. It cannot be overwritten by setting a new value in **gc_SetUserInfo( )** or **gc_MakeCall( )**.

Table 28. Parameters Configurable Using gc_SetConfigData( ) When Using SIP (Continued)

Set ID	Parameter IDs	Use Before†
IPSET_DTMF	IPPARM_SUPPORT_DTMF_BITMASK Datatype: Uint8[ ] IPPARM_DTMF_RFC2833_PAYLOAD_TYPE Datatype: Uint8[ ]	**gc_AnswerCall( )** **gc_MakeCall( )**
IPSET_EXTENSIONEVT_MSK	GCACT_ADDMSK Datatype: Uint8[ ] GCACT_SETMSK Datatype: Uint8[ ] GCACT_SUBMSK Datatype: Uint8[ ]	**gc_AnswerCall( )**
† Information can be set in any state but it is only used in certain states. See Section 7.1, "Global Call Functions Supported by IP", on page 153 for more information. †† This is a system configuration parameter for the terminating side, not a call configuration parameter. It cannot be overwritten by setting a new value in **gc_SetUserInfo( )** or **gc_MakeCall( )**.		

7.2.25 gc_SetUserInfo( ) Variances for IP

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

- set call values for all calls on the specified line device
- set call values for the duration of a single call
- set SIP message information fields
- associate and disassociate a T.38 Fax device with a Media device

The **gc_SetUserInfo**( ) function is used only to set the values of call-related information, such as coder information, display information, phone list, etc. before a call has been initiated. The information is not transmitted until the next Global Call function that initiates the transmission of information on the line, such as, **gc_AnswerCall**( ), **gc_AcceptCall**( ), or **gc_CallAck**( ).

Note: If no coder type is specified, all supported coder types will be used in the coder negotiation.

The parameters that are configurable using **gc_SetUserInfo**( ) are given in Table 21, "Configurable Call Parameters When Using H.323", on page 173 and Table 22, "Configurable Call Parameters When Using SIP", on page 176. In addition, the DTMF support bitmask, (see Table 27 and Table 28) is also configurable using **gc_SetUserInfo**( ).

Note: The **gc_SetUserInfo**( ) function may **not** be used to set the IP protocol for a multi-protocol line device (i.e., one that was opened in P_IP mode). The only mechanism for selecting the protocol to use is the GC_MAKECALL_BLK structure associated with the **gc_MakeCall**( ) function.

The **gc_SetUserInfo**( ) function operates on either a CRN or a line device:

- If the target of the function is a CRN, the information in the function is automatically directed to the protocol associated with that CRN.
- If the target of the function is a line device, then:
  - If the line device was opened as a multi-protocol device (:N_PIP), the information in the function is automatically directed to each protocol and is used by either H.323 or SIP calls made subsequently.

– If the line device was opened as a single-protocol device (:N_H323 or :N_SIP), then the information in the function automatically applies to that protocol only and is used by calls made using that protocol.

Note: Use **gc_SetConfigData( )** to set parameters on board devices.

The **gc_SetUserInfo( )** is also used to set Information Elements (IEs) in Q.931 messages. See Section 4.5.3, "Setting Q.931 Message IEs", on page 90 for more information.

7.2.25.1 Setting Call Parameters for the Next Call

The relevant function parameter values in this context are:

target_type
: GCTGT_GCLIB_CRN (if a CRN exists) or GCTGT_GCLIB_CHAN (if a CRN does not exist)

target_id
: CRN (if it exists) or line device (if a CRN does not exist)

duration
: GC_SINGLECALL

infoparmblkp
: a pointer to a GC_PARM_BLK with a list of parameters (including coder information) to be set for the line device.

Note: If a call is in the Null state, the new parameter values apply to the next call. If a call is in a non-Null state, the new parameter values apply to the remainder of the current call only.

7.2.25.2 Setting Call Parameters for the Next and Subsequent Calls

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
: line device

duration
: GC_ALLCALLS

infoparmblkp
: a pointer to a GC_PARM_BLK with a list of parameters (including coder information) to be set for the line device.

Note: If a call is in the Null state, the new parameter values apply to the next call and all subsequent calls. If a call is in a non-Null state, the new parameter values apply to the remainder of the current call and all subsequent calls.

7.2.25.3 Setting SIP Message Information Fields

The **gc_SetUserInfo( )** function can be used to set SIP message information fields. The relevant function parameter values in this context are:

target_type
GCTGT_GCLIB_CHAN

target_id
line device

duration
GC_SINGLECALL

infoparmblkp
A pointer to a GC_PARM_BLK that contains the IPSET_SIP_MSGINFO parameter set ID and one of the following parameter IDs that identify the fields to be set:

- IPPARM_CALLID_HDR
- IPPARM_CONTACT_DISPLAY
- IPPARM_CONTACT_URI
- IPPARM_DIVERSION_URI
- IPPARM_FROM_DISPLAY
- IPPARM_REFERRED_BY
- IPPARM_REPLACES
- IPPARM_REQUEST_URI
- IPPARM_TO_DISPLAY

See Section 4.6.3, "Setting a SIP Message Information Field", on page 93 for more information and a cod e example.

7.2.25.4 Associating and Disassociating a T.38 Fax Device with a Media Device

To support T.38 fax server operation, the **gc_SetUserInfo( )** function is used to associate a T.38 Fax device with a Media device to facilitate a switch from an audio session to a T.38 fax session. Similarly, when switching form a T.38 fax session to an audio session, the **gc_SetUserInfo( )** function is used to disassociate the T.38 Fax device from the Media device. The relevant function parameter values in this context are:

target_type
GCTGT_GCLIB_CRN

target_id
CRN

duration
GC_SINGLECALL

infoparmblkp
a pointer to a GC_PARM_BLK that contains:

- the IPSET_FOIP parameter set ID and one of the following parameter IDs:
  – IPPARM_T38_CONNECT when switching from audio to T.38 fax
  – IPPARM_T38_DISCONNECT when switching from T.38 fax to audio
- an associated IP_CONNECT structure that contains the fax and media handles and the connection type (half-duplex or full-duplex)

See Section 4.18.3, “Initiating a Switch from Audio to T.38 Fax”, on page 128 for more information and a code example.

7.2.26 gc_Start( ) Variances for IP

The **gc_Start( )** function is used to define the number of IPT board devices to create (see Section 2.3.2, “IPT Board Devices”, on page 35 for the meaning of an IPT board device) and the parameters for each IPT board device.

The number of IPT boards is identified in an IPCCLIB_START_DATA structure that also contains a pointer to an array of IP_VIRTBOARD structures (one structure for each board) that contain the parameters for each board. The parameters include:

- default address string delimiter character
- total number of IPT devices that can be open concurrently
- maximum number of IPT devices to be used for H.323 calls
- H.323 local address and signaling port
- maximum number of IPT devices to be used for SIP calls
- SIP local address and signaling port
- enable/disable access to SIP message information fields
- enable/disable call transfer functionality when using H.450.2
- enable/disable access to H.323 message information fields
- a parameter to set the Terminal Type that will be set when the board is started

See reference page for IP_VIRTBOARD on page 243 for more information.

Two convenience functions, **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** (defined in the *gcip.h* header file) **must** be used to initialize the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default settings. The default settings can then be overridden by desired values. For example, the application may need to set specific values in IP_VIRTBOARD to override the default value for terminal type, to enable access to H.323 message information fields, to enable call transfer services for the H.450.2 protocol, or to enable access to SIP message information fields.

The following code example illustrates the use of these convenience functions:

```
IP_VIRTBOARD ip_virtboard[2];
IPCCLIB_START_DATA ipcclibstart;
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].sup_serv_mask = IP_SUP_SERV_CALL_XFER; /* override supp services default */
ip_virtboard[1].sup_serv_mask = IP_SUP_SERV_CALL_XFER; /* override supp services default */
```

If NULL is passed to **gc_Start( )**, the function will create and initialize two virtual boards, one for H.323 and one for SIP, using default parameter values.

The following parameter applies to both boards:

- delimiter = , [default parsing delimiter for address strings is a comma]

The parameters for the default H.323 virtual board are:

- total_max_calls = 120
- h323_max_calls = 120
- h323_signaling_port = 1720
- localIP.ip_ver = IPVER4
- localIP.u_ipaddr.ipv4 — determined by socket functions
- h323_msginfo_mask = IP_H323_MSGINFO_DISABLE
- sup_serv_mask = IP_SUP_SERV_DISABLED
- terminal_type = IP_TT_GATEWAY

The parameters for the default SIP virtual board are:

- total_max_calls = 120
- sip_max_calls = 120
- sip_signaling_port = 5060
- localIP.ip_ver = IPVER4
- localIP.u_ipaddr.ipv4 — determined by socket functions
- sip_msg_info_mask = IP_SIP_MSGINFO_DISABLE
- sup_serv_mask = IP_SUP_SERV_DISABLED
- outbound_proxy_IP = Disabled
- outbound_proxy_port = 5060
- outbound_proxy_hostname = NULL
- terminal_type = N/A (not used)

The total number of IPT devices is not necessarily the number of IPT devices used for H.323 calls plus the number of IPT devices used for SIP calls. Each IPT device can be used for both H.323 and SIP. If there are 2016 devices available (total_max_calls=2016, three Intel NetStructure IPT boards), you can specify that all 2016 devices can be used for both H.323 calls (max_h323=2016) and SIP (max_sip=2016), or half are used for H.323 only (max_h323=1008) and half are used for SIP only (max_sip=1008), or any other such combination.

The default value for the maximum number of IPT devices is 120, but this can be set to a value up to 2016. See the reference page for IP_VIRTBOARD on page 243 for more information. The local IP address for each IPT board device is a parameter of type IPADDR in the IP_VIRTBOARD structure. See the reference page for IPADDR on page 245 for more information.

Notes: ***1.*** When using Global Gall over IP, the GC_LIB_START structure must include both the GC_H3R_LIB and GC_IPM_LIB libraries since there are inter-dependencies.

2. The maximum value of the num_boards field is 8.

The total_max_calls, h323_max_calls, and SIP_max_calls fields in the IP_VIRTBOARD structure can be used to allocate the number and types of calls among the available devices. The following #defines have been provided as a convenience to the application developer:

IP_CFG_DEFAULT
: indicates to the call control library that it should determine and fill in the correct value

IP_CFG_MAX_AVAILABLE_CALLS
: indicates to the call control library that it should use the maximum available resources

Note: Do not use IP_CFG_MAX_AVAILABLE_CALLS with applications running on HMP 1.1, since IP_CFG_MAX_AVAILABLE_CALLS will initialize the stack for 2016 channels. This is an inefficient use of system resources, resulting in a lengthy initialization time and substantial memory use.

IP_CFG_NO_CALLS
: indicates to the call control library that it should **not** allocate any resources

The following restrictions apply when overriding values in the IPCCLIB_START_DATA structure. The **gc_Start( )** function will fail if these restrictions are not observed.

- The total number of devices (total_max_calls) must not be larger than the sum of the values for the maximum number of H.323 calls (h323_max_calls) and the maximum number of SIP calls (sip_max_calls).
- The total number of devices (total_max_calls) cannot be set to IP_CFG_NO_CALLS.
- The maximum number of H.323 calls (h323_max_calls) and maximum number of SIP calls (sip_max_calls) values cannot both be set to IP_CFG_NO_CALLS.
- When configuring multiple board devices, IP_CFG_DEFAULT cannot be used as an address specifier.
- If different IP addresses or port numbers are not used when running multiple instances of an application for any one technology (H.323 or SIP), then the xxx_max_calls (xxx = h323 or sip) parameter for the other technology must be set to IP_CFG_NO_CALLS.

7.2.27 gc_UnListen( ) Variances for IP

The **gc_UnListen( )** function is supported in both asynchronous and synchronous modes. The function is blocking in synchronous mode.

Note: For line devices that comprise media (ipm) and voice (dxxx) devices, routing is only done on the media devices. Routing of the voice devices must be done using the Voice API (dx_ functions).

7.3 Global Call States Supported by IP

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

- GCST_ACCEPTED

- GCST_ACCEPT_XFER
- GCST_ALERTING
- GCST_CALLROUTING
- GCST_CONNECTED
- GCST_DETECTED
- GCST_DIALING
- GCST_DISCONNECTED
- GCST_IDLE
- GCST_INVOKE_XFER_ACCEPTED
- GCST_INVOKE_XFER
- GCST_NULL
- GCST_OFFERED
- GCST_PROCEEDING
- GCST_REQ_INIT_XFER
- GCST_REQ_XFER
- GCST_XFER_CMPLT

See the *Global Call API Programming Guide* for more information about the call state models.

7.4 Global Call Events Supported by IP

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

- GCEV_ACCEPT
- GCEV_ACCEPT_INIT_XFER
- GCEV_ACCEPT_INIT_XFER_FAIL
- GCEV_ACCEPT_XFER
- GCEV_ACCEPT_XFER_FAIL
- GCEV_ACKCALL (deprecated; equivalent is GCEV_CALLPROC)
- GCEV_ALARM
- GCEV_ALERTING
- GCEV_ANSWERED
- GCEV_ATTACH
- GCEV_ATTACHFAIL
- GCEV_BLOCKED
- GCEV_CONNECTED
- GCEV_CALLPROC
- GCEV_DETECTED
- GCEV_DETACH

- GCEV_DETACHFAIL
- GCEV_DIALING
- GCEV_DISCONNECTED
- GCEV_DROPCALL
- GCEV_ERROR
- GCEV_EXTENSION (unsolicited event)
- GCEV_EXTENSIONCMPLT (termination event for **gc_Extension( )**)
- GCEV_FATALERROR
- GCEV_INIT_XFER
- GCEV_INIT_XFER_FAIL
- GCEV_INIT_XFER_REJ
- GCEV_INVOKE_XFER
- GCEV_INVOKE_XFER_FAIL
- GCEV_INVOKE_XFER_REJ
- GCEV_LISTEN
- GCEV_OFFERED
- GCEV_OPENEX
- GCEV_OPENEX_FAIL
- GCEV_PROCEEDING
- GCEV_REJ_INIT_XFER
- GCEV_REJ_INIT_XFER_FAIL
- GCEV_REJ_XFER
- GCEV_REJ_XFER_FAIL
- GCEV_RELEASECALL
- GCEV_REQ_INIT_XFER
- GCEV_REQ_XFER
- GCEV_RESETLINEDEV
- GCEV_SERVICEREQ
- GCEV_SERVICERESP
- GCEV_SERVICERESPCMPLT
- GCEV_SETCONFIGDATA
- GCEV_SETCONFIGDATAFAIL
- GCEV_TASKFAIL
- GCEV_UNBLOCKED
- GCEV_UNLISTEN
- GCEV_XFER_CMPLT
- GCEV_XFER_FAIL

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

7.5 Initialization Functions

Two initialization functions, defined as inline functions in the *gcip.h* header file, provide the mechanism for initializing startup and IPT board structures.

7.5.1 INIT_IPCCLIB_START_DATA( )

The function prototype is defined as follows:

```
void INIT_IPCCLIB_START_DATA(IPCCLIB_START_DATA *pIpStData,
                             unsigned char numBoards,
                             IP_VIRTBOARD *pIpVb)
```

Applications **must** use this function to initialize the IPCCLIB_START_DATA structure before calling **gc_Start( )**.

The function takes the following parameters:

pIpStData
: a pointer to the IPCCLIB_START_DATA structure to be initialized

numBoards
: the number of virtual IPT boards being defined (up to a maximum of 8)

pIpVb
: a pointer to an array of IP_VIRTBOARD structures, one for each IPT board

This function is used when using the **gc_Start( )** function. See Section 7.2.26, “gc_Start( ) Variances for IP”, on page 201 for a code example of how to use this function.

7.5.2 INIT_IP_VIRTBOARD( )

The function prototype is defined as follows:

```
void INIT_IP_VIRTBOARD(IP_VIRTBOARD *pIpVb)
```

Applications **must** use this function to initialize the IP_VIRTBOARD structure associated with each IPT board device before calling **gc_Start( )**. The function sets IP_VIRTBOARD fields to default values; see the reference page for IP_VIRTBOARD on page 243 for more information. The application can then override these defaults as desired. In particular, the application may need to set specific values in IP_VIRTBOARD to override the default value for terminal type (which is Gateway), to enable access to H.323 message information fields, to enable call transfer services for the H.450.2 protocol, or to enable access to SIP message information fields.

The function takes one parameter:

pIpVb
: a pointer to the IP_VIRTBOARD structure for a specific IPT board device.

This function is used when using the **gc_Start( )** function. See Section 7.2.26, "gc_Start( ) Variances for IP", on page 201 for a code example of how to use this function.

IP-Specific Parameters 8

This chapter describes the parameter set IDs (set IDs) and parameter IDs (parm IDs) used with IP technology. Topics include:

8.1 Overview of Parameter Usage

The parameter set IDs and parameter IDs described in this chapter are defined in the *gcip.h* header file. Table 29 summarizes the parameter sets and parameters used by Global Call in an IP environment, organized alphabetically by set ID and then by parameter ID.

The meaning of the columns in Table 29 are:

- **Set ID** – An identifier for a group of related parameters.
- **Parameter ID** – An identifier for a specific parameter.
- **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 endpoint.
- **Retrieve** – Indicates the Global Call function used to retrieve information that was sent by a peer endpoint.
- **H.323/SIP** – Indicates if the parameter is supported when using H.323, SIP, or both.

Detailed information about each of the parameters in each parameter set is provided in the second part of this chapter.

Table 29. Summary of Parameter Sets and Parameter Usage

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
GCSET_ CALL_CONFIG	GCPARM_ CALLPROC	**gc_SetConfigData( )**	---	---	both
GCSET_ CHAN_ CAPABILITY	IPPARM_ LOCAL_CAPABILITY	**gc_SetConfigData( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	both
IPSET_ CALLINFO	IPPARM_ BEARERCAP	**gc_SetUserInfo( )** (GC_SINGLECALL only)	**gc_MakeCall( )**	Retrieve from GCEV_OFFERED event via **gc_GetMetaEvent( )**	H.323 only
	IPPARM_ CALLDURATION	---	---	**gc_Extension( )** (IPEXTID_GETINFO)	both

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ CALLINFO	IPPARM_CALLID	**gc_MakeCall( )** **gc_SetUserInfo( )** (GC_SINGLECALL only)	**gc_MakeCall( )**	**gc_GetCallInfo( )** (IP_CALLID) –or– **gc_Extension( )** (IPEXTID_GETINFO) **Note:** The use of **gc_Extension( )** to retrieve the Call ID is being deprecated; use **gc_GetCallInfo( )** instead.	both
	IPPARM_ CONNECTION METHOD	**gc_MakeCall( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	both
	IPPARM_DISPLAY	**gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	both
	IPPARM_FACILITY	**gc_SetUserInfo( )** (GC_SINGLECALL only)	**gc_AnswerCall( )** **gc_MakeCall( )**	In GCEV_OFFERED, GCEV_CONNECTED, or GCEV_EXTENSION event (with ext_id of IPEXTID_ RECEIVEMSG). Retrieve IE value with **gc_GetMetaEvent( )**	H.323 only
	IPPARM_ H245TUNNELING	**gc_SetUserInfo( )** † **gc_MakeCall( )** **gc_SetConfigData( )** ‡	**gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_MEDIA WAITFORCONNECT	**gc_SetUserInfo( )**	**gc_MakeCall( )**	**gc_GetMetaEvent( )** (GCEV_OFFERED) **gc_util_next_parm( )**	H.323 only
	IPPARM_ PHONELIST	**gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	both
	IPPARM_ PRESENTATION_IND	**gc_SetUserInfo( )**	**gc_MakeCall( )**	**gc_GetMetaEvent( )** (GCEV_OFFERED) **gc_util_next_parm( )**	H.323 only

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ CALLINFO	IPPARM_ PROGRESS_IND	---	---	**gc_GetMetaEvent( )** (GCEV_EXTENSION) **gc_util_next_parm( )** **Note:** Extension events associated with Progress messages are masked by default.Enable them via gc_SetUserInfo(IPSET_EXTENSIONEVT_MSK, GCACT_SETMSK, EXTENSIOEVT_ CALL_PROGRESS)	H.323 only
	IPPARM_ USERUSER_INFO	**gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
IPSET_ CONFERENCE	IPPARM_ CONFERENCE_ GOAL	**gc_MakeCall( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ CONFERENCE_ID	---	---	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
IPSET_CONFIG	IPPARM_ CONFIG_TOS	**gc_MakeCall( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	both
	IPPARM_ OPERATING_MODE	**gc_SetConfigData( )**	---	---	both
IPSET_DTMF	IPPARM_ DTMF_ ALPHANUMERIC	---	**gc_Extension( )** (IPEXTID_SEND_ DTMF)	**gc_Extension( )** (IPEXTID_ RECEIVE_DTMF)	both
	IPPARM_ DTMF_RFC2833_ PAYLOAD_TYPE	**gc_SetConfigData( )** **gc_SetUserInfo( )** †	---	---	both
	IPPARM_ SUPPORT_DTMF_ BITMASK	**gc_SetConfigData( )** **gc_SetUserInfo( )** †	---	---	both
IPSET_ EXTENSIONEVT_ MSK	GCACT_ADDMSK	**gc_SetConfigData( )**	---	---	both
	GCACT_GET_MSK	**gc_SetConfigData( )**	---	---	both
	GCACT_SETMSK	**gc_SetConfigData( )**	---	---	both
	GCACT_SUBMSK	**gc_SetConfigData( )**	---	---	both

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_FOIP	IPPARM_ T38_OFFERED	---	---	GCEV_OFFERED	both
	IPPARM_ T38_CONNECT	**gc_SetUserInfo( )**	---	---	both
	IPPARM_ T38_DISCONNECT	**gc_SetUserInfo( )**	---	---	both
IPSET_ H323_ RESPONSE_ CODE	IPPARM_BUSY_ CAUSE	**gc_SetConfigData( )**	---	---	H.323 only
IPSET_ IPPROTOCOL_ STATE	IPPARM_ CONTROL_ CONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ IPPROTOCOL_ STATE)	H.323 only
IPSET_ IPPROTOCOL_ STATE	IPPARM_ CONTROL_ DISCONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ IPPROTOCOL_ STATE)	H.323 only
	IPPARM_ SIGNALING_ CONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ IPPROTOCOL_ STATE)	H.323 only
	IPPARM_ SIGNALING_ DISCONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ IPPROTOCOL_ STATE)	H.323 only
IPSET_ LOCAL_ALIAS	IPPARM_ ADDRESS_DOT_ NOTATION	---	**gc_ReqService( )**	---	both
	IPPARM_ ADDRESS_EMAIL	---	**gc_ReqService( )**	---	both
	IPPARM_ ADDRESS_H323_ID	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_PHONE	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_ TRANSPARENT	---	**gc_ReqService( )**	---	both
	IPPARM_ ADDRESS_URL	---	**gc_ReqService( )**	---	H.323 only
IPSET_ MEDIA_STATE	IPPARM_ RX_CONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ MEDIAINFO)	both

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ MEDIA_STATE	IPPARM_ RX_ DISCONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ MEDIAINFO)	both
	IPPARM_ TX_CONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ MEDIAINFO)	both
	IPPARM_ TX_ DISCONNECTED	---	---	GCEV_EXTENSION (IPEXTID_ MEDIAINFO)	both
IPSET_ MSG_H245	IPPARM_MSGTYPE	---	**gc_Extension( )** (IPEXTID_ SENDMSG)	GCEV_EXTENSION (IPEXTID_ RECEIVEMSG)	H.323 only
IPSET_ MSG_Q931	IPPARM_MSGTYPE	---	**gc_Extension( )** (IPEXTID_ SENDMSG)	GCEV_EXTENSION (IPEXTID_ RECEIVEMSG)	H.323 only
IPSET_ MSG_ REGISTRATION	IPPARM_MSGTYPE	---	**gc_Extension( )** (IPEXTID_ SENDMSG)	GCEV_EXTENSION (IPEXTID_ RECEIVEMSG)	both
IPSET_ NONSTANDARD CONTROL	IPPARM_ H221NON STANDARD	**gc_SetConfigData( )** **gc_MakeCall( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ NONSTANDARD DATA_DATA	**gc_SetConfigData( )** **gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_AnswerCall( )** **gc_MakeCall( )** **gc_DropCall( )** **gc_ReqService( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ NONSTANDARD DATA_OBJID	**gc_SetConfigData( )** **gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_AnswerCall( )** **gc_MakeCall( )** **gc_DropCall( )** **gc_ReqService( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
IPSET_ NONSTANDARD DATA	IPPARM_ H221NON STANDARD	**gc_SetConfigData( )** **gc_MakeCall( )** **gc_SetUserInfo( )** †	**gc_AnswerCall( )** **gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ NONSTANDARD DATA_DATA	**gc_SetConfigData( )** **gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_AnswerCall( )** **gc_MakeCall( )** **gc_DropCall( )** **gc_ReqService( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ NONSTANDARD DATA_OBJID	**gc_SetConfigData( )** **gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_AnswerCall( )** **gc_MakeCall( )** **gc_DropCall( )** **gc_ReqService( )**	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
IPSET_ PROTOCOL	IPPARM_ PROTOCOL_ BITMASK	**gc_SetConfigData( )** **gc_SetUserInfo( )** † **gc_MakeCall( )**	**gc_ReqService( )** **gc_MakeCall( )**	---	both

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ REG_INFO	IPPARM_ OPERATION_ DEREGISTER	---	**gc_ReqService( )**	---	both
	IPPARM_ OPERATION_ REGISTER	---	**gc_ReqService( )**	---	both
	IPPARM_ REG_ADDRESS		**gc_ReqService( )**	---	both
	IPPARM_ REG_STATUS	---	---	Forwarded automatically in a GCEV_SERVICERESP event	both
IPSET_SIP_ MSGINFO	IPPARM_ CALLID_HDR	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ CONTACT_DISPLAY	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ CONTACT_URI	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ DIVERSION_URI	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ FROM_DISPLAY	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ REFERRED_BY	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ REPLACES	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ REQUEST_URI	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
	IPPARM_ TO_DISPLAY	**gc_SetUserInfo( )** (GC_SINGLECALL) **gc_MakeCall( )**	**gc_MakeCall( )**	From GCEV_ OFFERED event	SIP only
IPSET_ SIP_ RESPONSE_ CODE	IPPARM_BUSY_ REASON	**gc_SetConfigData( )**	---	---	SIP only

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ SUPPORTED_ PREFIXES	IPPARM_ ADDRESS_DOT_ NOTATION	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_EMAIL	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_ H323_ID	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_PHONE	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_ TRANSPARENT	---	**gc_ReqService( )**	---	H.323 only
	IPPARM_ ADDRESS_URL	---	**gc_ReqService( )**	---	H.323 only
IPSET_ SWITCH_ CODEC	IPPARM_ACCEPT	---	**gc_Extension( )** (IPEXTID_ CHANGE_MODE)	---	both
	IPPARM_ AUDIO_INITIATE	---	**gc_Extension( )** (IPEXTID_ CHANGE_MODE)	---	both
	IPPARM_ AUDIO_ REQUESTED	---	---	GCEV_EXTENSION (IPEXTID_ CHANGE_MODE)	both
	IPPARM_READY	---	---	GCEV_EXTENSION (IPEXTID_ CHANGE_MODE)	both
	IPPARM_REJECT	---	**gc_Extension( )** (IPEXTID_ CHANGE_MODE)	---	both
	IPPARM_ T38_INITIATE	---	**gc_Extension( )** (IPEXTID_ CHANGE_MODE)	---	both
	IPPARM_ T38_REQUESTED	---	---	GCEV_EXTENSION (IPEXTID_ CHANGE_MODE)	both
IPSET_ TRANSACTION	IPPARM_ TRANSACTION_ID	---	---	**gc_Extension( )** (Any ext_id)	both

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

Table 29. Summary of Parameter Sets and Parameter Usage (Continued)

Set ID	Parameter ID	Set	Send	Retrieve	SIP/ H.323
IPSET_ VENDORINFO	IPPARM_ H221NONSTD	**gc_SetConfigData( )**	**gc_Extension( )** (IPEXTID_ SENDMSG)	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ VENDOR_ PRODUCT_ID	**gc_SetConfigData( )**	**gc_Extension( )** (IPEXTID_ SENDMSG)	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only
	IPPARM_ VENDOR_ VERSION_ID	**gc_SetConfigData( )**	**gc_Extension( )** (IPEXTID_ SENDMSG)	**gc_Extension( )** (IPEXTID_GETINFO)	H.323 only

† The **duration** parameter can be set to GC_SINGLECALL (to apply on a call basis) or to GC_ALLCALLS (to apply on a line device basis).
‡ Tunneling for incoming calls can only be specified using the **gc_SetConfigData( )** function with a board device target ID.

8.2 Parameter Set Reference

This section contains reference information on the parameters in each parameter set used for IP telephony under Global Call. The table in each of the following subsections lists and describes the individual parameters associated with the parameter set as well as indicating the data type, size, and defined values for the parameters.

The parameter sets documented in this section include:

- GCSET_CALL_CONFIG
- IPSET_CALLINFO
- IPSET_CONFERENCE
- IPSET_CONFIG
- IPSET_DTMF
- IPSET_EXTENSIONEVT_MSK
- IPSET_FOIP
- IPSET_H323_RESPONSE_CODE
- IPSET_IPPROTOCOL_STATE
- IPSET_LOCAL_ALIAS
- IPSET_MEDIA_STATE
- IPSET_MSG_H245
- IPSET_MSG_Q931
- IPSET_MSG_REGISTRATION
- IPSET_NONSTANDARDCONTROL
- IPSET_NONSTANDARDDATA
- IPSET_PROTOCOL
- IPSET_REG_INFO

- IPSET_SIP_MSGINFO
- IPSET_SIP_RESPONSE_CODE
- IPSET_SUPPORTED_PREFIXES
- IPSET_SWITCH_CODEC
- IPSET_TRANSACTION
- IPSET_VENDORINFO

8.2.1 GCSET_CALL_CONFIG

Table 30 shows the parameter IDs in the GCSET_CALL_CONFIG parameter set that are relevant in an IP context.

Table 30. GCSET_CALL_CONFIG Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
GCPARM_CALLPROC	Type: enumeration Size: sizeof(char) Values: • GCCONTROL_APP - The application must use **gc_CallAck( )** to send the Proceeding message. This is the default. • GCCONTROL_TCCL - The stack sends the Proceeding message automatically.	Used to specify if the Proceeding message is sent under application control or automatically by the stack	both

8.2.2 IPSET_CALLINFO

Table 31 shows the parameter IDs in the IPSET_CALLINFO parameter set.

Table 31. IPSET_CALLINFO Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_BEARERCAP	Type: string Size: max. length = 255	Bearer Capability IE	H.323 only
IPPARM_CALLDURATION	Type: unsigned int Size: sizeof(unsigned int)	Duration of the call	H.323 only
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.			

Table 31. IPSET_CALLINFO Parameter Set (Continued)

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_CALLID	Type for SIP: string Size for SIP: max. length = MAX_IP_SIP_CALLID_LENGTH Type for H.323: array of octets Sixe for H.323: MAX_IP_H323_ CALLID_LENGTH If protocol is unknown, MAX_IP_ CALLID_LENGTH defines the the maximum Call ID length for any supported protocol.	Globally unique identifier (Call ID) used by the underlying protocol to identify the call **Note:** When using SIP, direct manipulation of the Call ID message header via IPSET_SIP_MSGINFO / IPPARM_CALLID_HDR overrides any value provided via this parameter.	both
IPPARM_ CONNECTIONMETHOD	Type: enumeration Size: sizeof(char) Values: • IP_CONNECTIONMETHOD_ FASTSTART • IP_CONNECTIONMETHOD_ SLOWSTART	The connection method: Fast Start or Slow Start. See Section 4.2, "Using Fast Start and Slow Start Setup", on page 66 for more information.	both
IPPARM_DISPLAY	Type: string Size: max. length = MAX_DISPLAY_LENGTH (82), null-terminated	Display information. This information can be used by a peer as additional address information.	both
IPPARM_FACILITY	Type: string Size: max. length = 255	Facility IE associated with SETUP, CONNECT, or FACILITY message. A Global Call Extension ID of EXTID_RECEIVEMSG applies when the IE is in an incoming FACILITY message.	H.323 only
IPPARM_H245TUNNELING	Type: enumeration Size: sizeof(char) Values: • IP_H245TUNNELING_ON • IP_H245TUNNELING_OFF	Specify if tunneling is on or off. See Section 4.13, "Enabling and Disabling Tunneling in H.323", on page 107 for more information.	H.323 only
IPPARM_ MEDIAWAITFORCONNECT	Size: sizeof(char) Values: • 0 = FALSE • 1 = TRUE	MediaWaitForConnect field in SETUP message.	H.323 only

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.

Table 31. IPSET_CALLINFO Parameter Set (Continued)

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_PHONELIST	Type: string Size: max. length = MAX_ADDRESS_LENGTH (128)	Phone numbers that can be retrieved at the remote end point. **Note:** When issuing a **gc_MakeCall( )**, this information can also be sent through the **numberstr** parameter. See Section 7.2.16, "gc_MakeCall( ) Variances for IP", on page 172 for more information.	both
IPPARM_ PRESENTATION_IND	Type: enumeration Size: sizeof(char) Values: • IP_PRESENTATIONALLOWED • IP_PRESENTATION RESTRICTED	PresentationIndicator field in incoming and outgoing SETUP messages. An application may use this field to control whether the Caller ID is presented to the user.	H.323 only
IPPARM_PROGRESS_IND	Type: string Size: max. length = 255	Data contained in the Progress Indicator IE in incoming PROGRESS messages. **Note:** Extension events associated with PROGRESS messages are masked by default.Enable them with gc_SetUserInfo(IPSET_ EXTENSIONEVT_MSK, GCACT_SETMSK, EXTENSIOEVT_CALL_ PROGRESS)	H.323 only
IPPARM_USERUSER_INFO	Type: unsigned char[ ] Size: max size = MAX_USERUSER_INFO_ LENGTH (131)	User-to-user information	H.323 only
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.			

8.2.3 IPSET_CONFERENCE

Table 32 shows the parameter IDs in the IPSET_CONFERENCE parameter set.

Table 32. IPSET_CONFERENCE Parameter Set

Parameter ID	Type/Size	Description	SIP/ H.323
IPPARM_CONFERENCE_GOAL	Type: enumeration Size: sizeof(char) Values: • IP_CONFERENCEGOAL_UNDEFINED • IP_CONFERENCEGOAL_CREATE • IP_CONFERENCEGOAL_JOIN • IP_CONFERENCEGOAL_INVITE • IP_CONFERENCEGOAL_ CAP_NEGOTIATION • IP_CONFERENCEGOAL_ SUPPLEMENTARY_SRVC	The conference functionality to be achieved	H.323 only
IPPARM_CONFERENCE_ID	Type: string Size: max. length = IP_CONFERENCE_ID_LENGTH (16)	The conference identifier	H.323 only
1. 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. 2. Conference ID retrieval is only relevant when an application is in a conference. In a peer-to-peer call, the conference ID does not signify a call identifier. The application should use IPPARM_CALLID to retrieve the call identifier. See Section 8.2.2, "IPSET_CALLINFO", on page 217 for more information.			

8.2.4 IPSET_CONFIG

Table 33 shows the parameter IDs in the IPSET_CONFIG parameter set.

Table 33. IPSET_CONFIG Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_CONFIG_TOS	Type: char Size: sizeof(char)	Set the Type of Service (TOS) byte. Valid values are in the range 0 to 255. The default value is 0.	both
IPPARM_OPERATING_MODE	Type: int Size: sizeof(int)	Used when getting or setting the T.38 Fax Server mode. Possible values are: • IP_MANUAL_MODE	both

8.2.5 IPSET_DTMF

Table 34 shows the parameter IDs in the IPSET_DTMF parameter set. This parameter set is used to set DTMF-related parameters for the notification, suppression or sending of DTMF digits.

Table 34. IPSET_DTMF Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ DTMF_ALPHANUMERIC	Type: IP_DTMF_DIGITS Size: sizeof(IP_DTMF_DIGITS)	Used when sending or receiving DTMF via UII alphanumeric messages. The parameter value contains an IP_DTMF_DIGITS structure that includes the digit string.	both
IPPARM_ DTMF_RFC2833_ PAYLOAD_TYPE	Type: unsigned char Size: sizeof(char) Values: 96 to 127	Used to specify the RFC2833 RTP payload type. The default value is IP_USE_STANDARD_PAYLOADTYPE (101).	both
IPPARM_ SUPPORT_DTMF_ BITMASK	Type: int Size: sizeof(int)	Used to specify a bitmask that defines which DTMF transmission methods are to be supported. Possible values are: • IP_DTMF_TYPE_ALPHANUMERIC † • IP_DTMF_TYPE_INBAND_RTP • IP_DTMF_TYPE_RFC_2833	both

8.2.6 IPSET_EXTENSIONEVT_MSK

This parameter set is used to enable or disable the events associated with unsolicited notification such as the detection of DTMF or a change of connection state in an underlying protocol. Table 35 shows the parameter IDs in the IPSET_EXTENSIONEVT_MSK parameter set.

Table 35. IPSET_EXTENSIONEVT_MSK Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
GCPARM_GET_MSK	Type: int Size: sizeof(int)	Retrieve the bitmask of enabled events	both
GCACT_SETMSK	Type: int Size: sizeof(int)	Set the bitmask of enabled events.	both
GCACT_ADDMSK	Type: int Size: sizeof(int)	Add to the bitmask of enabled events	both

Table 35. IPSET_EXTENSIONEVT_MSK Parameter Set (Continued)

Parameter IDs	Type & Size	Description	SIP/ H.323
GCACT_SUBMSK	Type: int Size: sizeof(int)	Remove from the bitmask of enabled events	both
Values that can be used to make up the bitmask are: • EXTENSIONEVT_DTMF_ALPHANUMERIC (0x04) † • EXTENSIONEVT_SIGNALING_STATUS (0x08) • EXTENSIONEVT_STREAMING_STATUS (0x10) • EXTENSIONEVT_T38_STATUS (0x20)			

8.2.7 IPSET_FOIP

Table 36 shows the parameter IDs in the IPSET_FOIP parameter set.

Table 36. IPSET_FOIP Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_T38_OFFERED	Type: IP_CONNECT Size: sizeof(IP_CONNECT)	Used in a GC_PARM_BLK associated with an GCEV_OFFERED event to indicate that a T.38 session is requested.	both
IPPARM_T38_CONNECT	Type: IP_CONNECT Size: sizeof(IP_CONNECT)	Used when associating a T.38 Fax device with a Media device when switching from an audio session to a fax session.	both
IPPARM_T38_DISCONNECT	Type: IP_CONNECT Size: sizeof(IP_CONNECT)	Used when disassociating a T.38 Fax device with a Media device when switching from a fax session to an audio session.	both

8.2.8 IPSET_H323_RESPONSE_CODE

This parameter set is used to set the busy cause code that is used in the failure message sent when the local system is unable to accept additional incoming sessions.

Table 37. IPSET_H323_RESPONSE_CODE Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_BUSY_CAUSE	Type: eIP_EC_TYPE Size: sizeof(int)	Used in a GC_PARM_BLK to specify the cause code to send when no additional incoming sessions can be accepted. Values: • IPEC_Q931Cause34NoCircuitChannelAvailable • IPEC_Q931Cause47ResourceUnavailable Unspecified	H.323 only

8.2.9 IPSET_IPPROTOCOL_STATE

This parameter set is used when retrieving notification of protocol signaling states via GCEV_EXTENSION events. Table 38 shows the parameter IDs in the IPSET_IPPROTOCOL_STATE parameter set.

Table 38. IPSET_IPPROTOCOL_STATE Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ SIGNALING_CONNECTED	Type: int Size: sizeof(int)	Call signaling for the call has been established with the remote endpoint	H.323 only
IPPARM_ SIGNALING_DISCONNECTED	Type: int Size: sizeof(int)	Call signaling for the call has been terminated	H.323 only
IPPARM_ CONTROL_CONNECTED	Type: int Size: sizeof(int)	Media control signaling for the call has been established with the remote endpoint	H.323 only
IPPARM_ CONTROL_DISCONNECTED	Type: int Size: sizeof(int)	Media control signaling for the call has been terminated	H.323 only

8.2.10 IPSET_LOCAL_ALIAS

Table 39 shows the parameter IDs in the IPSET_LOCAL_ALIAS parameter set.

Table 39. IPSET_LOCAL_ALIAS Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ ADDRESS_DOT_NOTATION	Type: string Size: max. length = 255	A valid IP address	both
IPPARM_ ADDRESS_EMAIL	Type: string Size: max. length = 255	e-mail address composed of characters from the set "[A-Z][a-z][0-9]_-.@"	both
IPPARM_ ADDRESS_H323_ID	Type: string Size: max. length = 255	A valid H.323 ID	H.323 only
IPPARM_ ADDRESS_PHONE	Type: string Size: max. length = 255	An E.164 telephone number	H.323 only
IPPARM_ ADDRESS_TRANSPARENT	Type: string Size: max. length = 255	Unspecified address type	both
IPPARM_ ADDRESS_URL	Type: string Size: max. length = 255	A valid URL composed of characters from the set "[A-Z][a-z][0-9]-.". Must contain at least one "." and may not begin or end with a "-".	H.323 only

Note: For SIP, LOCAL_ALIAS is not the alias (or address of record), but rather the transport address or contact.

8.2.11 IPSET_MEDIA_STATE

Table 40 shows the parameter IDs in the IPSET_MEDIA_STATE parameter set.

Table 40. IPSET_MEDIA_STATE Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ RX_CONNECTED	Type: IP_CAPABILITY Size: sizeof(IP_CAPABILITY)	Streaming has been initiated in the receive direction from the remote endpoint. The IP_CAPABILITY structure includes coder information negotiated with the remote peer. See Section 4.11, “Enabling and Disabling Unsolicited Notification Events”, on page 105 for more information.	both
IPPARM_ RX_DISCONNECTED	Type: int Size: sizeof(int)	Streaming in the receive direction from the remote endpoint has been terminated.	both
IPPARM_ TX_CONNECTED	Type: IP_CAPABILITY Size: sizeof(IP_CAPABILITY)	Streaming has been initiated in the transmit direction toward the remote endpoint. The IP_CAPABILITY structure includes coder information negotiated with the remote peer. See Section 4.11, “Enabling and Disabling Unsolicited Notification Events”, on page 105 for more information.	both
IPPARM_ TX_DISCONNECTED	Type: int Size: sizeof(int)	Streaming in the transmit direction toward the remote endpoint has been terminated.	both

8.2.12 IPSET_MSG_H245

Table 41 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 41. IPSET_MSG_H245 Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_MSGTYPE	Type: int Size: sizeof(int)	Possible values for H.245 messages are: • IP_MSGTYPE_H245_INDICATION	H.323 only

8.2.13 IPSET_MSG_Q931

Table 42 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 or receive a Q.931 message.

Table 42. IPSET_MSG_Q931 Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_MSGTYPE	Type: int Size: sizeof(int)	Possible values for Q.931 messages are: • IP_MSGTYPE_Q931_FACILITY • IP_MSGTYPE_Q931_PROGRESS	H.323 only

8.2.14 IPSET_MSG_REGISTRATION

Table 43 shows the parameter IDs in the IPSET_MSG_REGISTRATION parameter set. This parameter set is used with the **gc_Extension( )** and the IPEXTID_SENDMSG extension and encapsulates all the parameters required to send a registration message. For information on the use of this parameter set, see Section 4.10.3, "Nonstandard Registration Message", on page 104.

Table 43. IPSET_MSG_REGISTRATION Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_MSGTYPE	Type: int Size: sizeof(int)	Possible value for registration messages is: • IP_MSGTYPE_REG_NONSTD	both

8.2.15 IPSET_NONSTANDARDCONTROL

Table 44 shows the parameter IDs in the IPSET_NONSTANDARDCONTROL parameter set.

Table 44. IPSET_NONSTANDARDCONTROL Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ NONSTANDARDDATA_DATA	Type: string Size: max. length = MAX_NS_PARM_ DATA_LENGTH (128)	Contains the nonstandard data supplied, if any. If nonstandard data was not supplied, this parameter should not be present in the parm block.	H.323 only
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.			

Table 44. IPSET_NONSTANDARDCONTROL Parameter Set (Continued)

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ NONSTANDARDDATA_OBJID	Type: Uint[ ] Size: max. length = MAX_NS_PARM_ OBJID_LENGTH (40)	Contains the nonstandard object ID supplied, if any. If a nonstandard object ID was not provided, this parameter should not be present in the parm block.	H.323 only
IPPARM_H221NONSTANDARD	Type: IP_H221NONSTANDARD Size: sizeof(IP_H221NONSTANDARD)	Contains an H.221 nonstandard data identifier.	H.323 only
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.			

8.2.16 IPSET_NONSTANDARDDATA

Table 45 shows the parameter IDs in the IPSET_NONSTANDARDDATA parameter set.

Table 45. IPSET_NONSTANDARDDATA Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ NONSTANDARDDATA_DATA	Type: string Size: max. length = MAX_NS_PARM_DATA_ LENGTH (128)	Contains the nonstandard data supplied, if any. If nonstandard data was not supplied, this parameter should not be present in the parm block.	H.323 only
IPPARM_ NONSTANDARDDATA_OBJID	Type: Uint[ ] Size: max. length = MAX_NS_PARM_OBJID_ LENGTH (40)	Contains the nonstandard object ID supplied, if any. If a nonstandard object ID was not provided, this parameter should not be present in the parm block.	H.323 only
IPPARM_H221NONSTANDARD	Type: IP_H221NONSTANDARD Size: sizeof(IP_H221NONSTANDARD)	Contains an H.221 nonstandard data identifier.	H.323 only
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.			

8.2.17 IPSET_PROTOCOL

Table 46 shows the parameter IDs in the IPSET_PROTOCOL parameter set.

Table 46. IPSET_PROTOCOL Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_PROTOCOL_BITMASK	Type: char Size: sizeof(char)	The IP protocol to use. Possible values are: • IP_PROTOCOL_H323 • IP_PROTOCOL_SIP	both

8.2.18 IPSET_REG_INFO

Table 47 shows the parameter IDs in the IPSET_REG_INFO parameter set.

Table 47. IPSET_REG_INFO Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ OPERATION_REGISTER	Type: char Size: sizeof(char)	Used to manipulate registration information when registering an endpoint with a gatekeeper/registrar. Possible values are: • IP_REG_ADD_INFO • IP_REG_DELETE_BY_VALUE • IP_REG_SET_INFO	both
IPPARM_ OPERATION_DEREGISTER	Type: char Size: sizeof(char)	Used when deregistering an endpoint with a gatekeeper/registrar. Possible values are: • IP_REG_DELETE_ALL – Discard the registration data in the local database • IP_REG_MAINTAIN_LOCAL_INFO – Keep the registration data in the local database	both
IPPARM_REG_ADDRESS	Type: IP_REGISTER_ ADDRESS Size: sizeof(IP_REGISTER_ ADDRESS)	Address information to be registered with a gatekeeper/registrar. See the reference page for IP_REGISTER_ADDRESS on page 242 for details.	both
IPPARM_REG_TYPE	Type: int Size: sizeof(int)	The registration type. Possible values are: • IP_REG_GATEWAY • IP_REG_TERMINAL	H.323 only
IPPARM_REG_STATUS	Type: char Size: sizeof(char)	Provides an indication of whether the endpoint registration with a gatekeeper/registrar was successful or not. Possible values are: • IP_REG_CONFIRMED • IP_REG_REJECTED	both

8.2.19 IPSET_SIP_MSGINFO

Table 48 shows the parameter IDs in the IPSET_SIP_MSGINFO parameter set. Note that access to SIP message info fields is disabled by default and must be explicitly enabled by activating the sip_msginfo_mask bitmask with the IP_SIP_MSGINFO_ENABLE mask value.

Table 48. IPSET_SIP_MSGINFO Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_CALLID_HDR	Type: string Max size: IP_CALLID_HDR_ MAXLEN	Used to set or retrieve the globally unique call identifier (Call ID) information from SIP messages. **Note:** Any value set via this parameter overrides any Call ID value set via IPSET_CALLINFO / IPPARM_CALLID.	SIP only
IPPARM_CONTACT_DISPLAY	Type: string Max size: IP_CONTACT_ DISPLAY_MAXLEN	Used to set or retrieve the Contact Display message information field in SIPINVITE messages	SIP only
IPPARM_CONTACT_URI	Type: string Max Size: IP_CONTACT_ URI_MAXLEN	Used to set or retrieve the Contact URI message information field from SIP INVITE messages	SIP only
IPPARM_DIVERSION_URI	Type: string Max Size: IP_DIVERSION_ URI_MAXLEN	Used to set or retrieve the Diversion URI message information field in SIP INVITE messages	SIP only
IPPARM_FROM_DISPLAY	Type: string Max Size: IP_FROM_ DISPLAY_MAXLEN	Used to set or retrieve the From Display message information field in SIP INVITE messages	SIP only
IPPARM_REFERRED_BY	Type: string Max Size: IP_REFERRED_ BY_MAXLEN	Used to set or retrieve the Referred-by message information field in SIP INVITE messages.	SIP only
IPPARM_REPLACES	Type: string Max Size: IP_REPLACES_ MAXLEN	Used to set or retrieve the Replaces message information field in SIP INVITE messages.	SIP only
IPPARM_REQUEST_URI	Type: string Max Size: IP_REQUEST_ URI_MAXLEN	Used to set or retrieve the Request URI message information field in SIP INVITE messages	SIP only
IPPARM_TO_DISPLAY	Type: string Max Size: IP_TO_DISPLAY_ MAXLEN	Used to set or retrieve the To Display message information field in SIP INVITE messages	SIP only

8.2.20 IPSET_SIP_RESPONSE_CODE

This parameter set is used to set the busy cause code that is used in the failure message sent when the local system is unable to accept additional incoming SIP sessions.

Table 49. IPSET_SIP_RESPONSE_CODE Parameter Set

Parameter ID	Type & Size	Description	SIP/ H.323
IPPARM_BUSY_REASON	Type: eIP_EC_TYPE Size: sizeof(int)	Used in a GC_PARM_BLK to specify the cause code to send when no additional incoming sessions can be accepted. Values: • IPEC_SIPReasonStatus480Temporarily Unavailable • IPEC_SIPReasonStatus486BusyHere • IPEC_SIPReasonStatus600BusyEverywhere	SIP only

8.2.21 IPSET_SUPPORTED_PREFIXES

Table 50 shows the parameter IDs in the IPSET_SUPPORTED_PREFIXES parameter set.

Table 50. IPSET_SUPPORTED_PREFIXES Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ ADDRESS_DOT_NOTATION	Type: string Size: max. length = 255	A valid IP address in dot notation	H.323 only
IPPARM_ ADDRESS_EMAIL	Type: string Size: max. length = 255	An e-mail address composed of characters from the set "[A-Z][a-z][0-9]_-.@"	H.323 only
IPPARM_ ADDRESS_H323_ID	Type: string Size: max. length = 255	A valid H.323 ID	H.323 only
IPPARM_ ADDRESS_PHONE	Type: string Size: max. length = 255	An E.164 telephone number	H.323 only
IPPARM_ ADDRESS_TRANSPARENT	Type: string Size: max. length = 255	Unspecified address type	H.323 only
IPPARM_ADDRESS_URL	Type: string Size: max. length = 255	A valid URL composed of characters from the set "[A-Z][a-z][0-9]-.". Must contain at least one "." and may not begin or end with a "-".	H.323 only

8.2.22 IPSET_SWITCH_CODEC

Table 51 shows the parameter IDs in the IPSET_SWITCH_CODEC parameter set.

Table 51. IPSET_SWITCH_CODEC Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_ACCEPT	Type: int Size: sizeof(int)	Used to accept an incoming coder switch request.	both
IPPARM_AUDIO_INITIATE	Type: int Size: sizeof(int)	Used to initiate the sending of a RequestMode (H.323) or REINVITE (SIP) message to the remote side to switch from T.38 fax to audio.	both
IPPARM_AUDIO_REQUESTED	Type: int Size: sizeof(int)	Provides notification of an incoming request to switch from T.38 fax to audio.	both
IPPARM_READY	Type: int Size: sizeof(int)	Provides notification that the media is ready.	both
IPPARM_REJECT	Type: int Size: sizeof(int)	Used to reject an incoming request to switch from audio to T.38 fax or vice versa.	both
IPPARM_T38_INITIATE	Type: int Size: sizeof(int)	Used to initiate the sending of a RequestMode (H.323) or REINVITE (SIP) message to the remote side to switch from audio to T.38 fax.	both
IPPARM_T38_REQUESTED	Type: int Size: sizeof(int)	Provides notification of an incoming request to switch from audio to T.38 fax.	both

8.2.23 IPSET_TRANSACTION

Table 52 shows the parameter IDs in the IPSET_TRANSACTION parameter set.

Table 52. IPSET_TRANSACTION Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_TRANSACTION_ID	Type: int Size: sizeof(int)	Used to uniquely identify any transaction	H.323 only

8.2.24 IPSET_VENDORINFO

Table 53 shows the parameter IDs in the IPSET_VENDORINFO parameter set.

Table 53. IPSET_VENDORINFO Parameter Set

Parameter IDs	Type & Size	Description	SIP/ H.323
IPPARM_H221NONSTD	Type: IP_H221NONSTANDARD Size: sizeof(IP_ H221NONSTANDARD)	Contains country code, extension code and manufacturer code. See the reference page for IP_H221NONSTANDARD on page 241 for details.	H.323 only
IPPARM_ VENDOR_PRODUCT_ID	Type: string Size: max. length = MAX_PRODUCT_ID_LENGTH (32)	Vendor product identifier	H.323 only
IPPARM_ VENDOR_VERSION_ID	Type: string Size: max. length = MAX_VERSION_ID_LENGTH (32)	Vendor version identifier	H.323 only
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.			

IP-Specific Data Structures 9

This chapter describes the data structures that are specific to IP technology.

Note: These data structures are defined in the *gcip.h* header file.

IP_AUDIO_CAPABILITY

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

■ Description

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

■ Field Descriptions

The fields of the IP_AUDIO_CAPABILITY data structure are described as follows:

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. This field can also be set to GCCAP_dontCare to indicate that any supported value is valid.

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 4.3.2, “Setting Coder Information”, on page 70 for more information.

VAD

Identifies whether voice activated detection (VAD) is enabled or disabled. Possible values are:

- GCPV_ENABLE – VAD enabled
- GCPV_DISABLE – VAD disabled
- GCCAP_dontCare – Any supported value is valid

IP_CAPABILITY

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

Description

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.

Field Descriptions

The fields of the IP_CAPABILITY data structure are described as follows:

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_NO_AUDIO
- GCCAP_DATA_t38UDPFax
- GCCAP_dontCare

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
- GCCAPTYPE_RDATA – Data

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.
- IP_CAP_DIR_LCLRXTX – Indicates a receive and transmit capability for the local endpoint. Supported for T.38 only.

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 nonstandard 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.

  Not currently supported.

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.

IP_CAPABILITY_UNION

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

Description

The IP_CAPABILITY_UNION union enables different capability categories to define their own additional parameters or interest.

Field Descriptions

The fields of the IP_CAPABILITY_UNION union are described as follows:

audio
: A structure that represents the audio capability. See IP_AUDIO_CAPABILITY, on page 234 for more information.

video
: Not supported.

data
: Not supported.

IP_CONNECT

```
typedef struct
{
   unsigned short     version;
   int                mediaHandle;
   int                faxHandle;
   eIPConnectType_e   connectType;
} IP_CONNECT;
```

■ Description

The IP_CONNECT data structure contains information required when associating a Media device with a T.38 Fax device required when switching from an audio coder to a T.38 coder and vice versa.

■ Field Descriptions

The fields of the IP_CONNECT data structure are described as follows:

version
: the current version number is 0x100

mediaHandle
: the Media device handle

faxHandle
: the T.38 Fax device handle

connectType
: the connection type. Possible values are:
  - IP_FULLDUP
  - IP_HALFDUP

Note: When disassociating a Media device from a T.38 Fax device, the faxHandle and connectType fields are ignored.

IP_DATA_CAPABILITY

```
typedef struct
{
   int     max_bit_rate;
} IP_DATA_CAPABILITY;
```

Description

The IP_DATA_CAPABILITY data structure provides additional information about the data capability.

Field Descriptions

The fields of the IP_DATA_CAPABILITY data structure are described as follows:

max_bit_rate

Possible values are:

- 2400
- 4800
- 9600
- 14400

The recommended value for T.38 coders is 14400.

IP_DTMF_DIGITS

```
typedef struct
{
    char          digit_buf[IP_MAX_DTMF_DIGITS];
    unsigned int  num_digits;
} IP_DTMF_DIGITS;
```

■ Description

The IP_DTMF_DIGITS data structure is used to provide DTMF information when the digits are received in a User Input Indication (UII) message with alphanumeric data.

■ Field Descriptions

The fields of the IP_DTMF_DIGITS data structure are described as follows:

digit_buf
: The DTMF digit string buffer; 32 characters in size

num_digits
: The number of DTMF digits in the string buffer

IP_H221NONSTANDARD

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

Description

The IP_H221NONSTANDARD data structure is used to store H.221 nonstandard data.

Field Descriptions

The fields of the IP_H221NONSTANDARD data structure are described as follows:

country_code
: The country code

extension
: The extension number

manufacturer_code
: The manufacturer code

IP_REGISTER_ADDRESS

```
typedef struct
{
    char            reg_client [IP_REG_CLIENT_ADDR_LENGTH];
    char            reg_server [IP_REG_SERVER_ADDR_LENGTH];
    int             time_to_live;
    int             max_hops;
} IP_REGISTER_ADDRESS;
```

Description

The IP_REGISTER_ADDRESS data structure is used to store registration information.

Field Descriptions

The fields of the IP_REGISTER_ADDRESS data structure are described as follows:

reg_client
: The meaning is protocol dependent:
  - When using H.323, this field is not used; any value specified is ignored
  - When using SIP, this field is an alias for the subscriber

reg_server
: The address of the registration server. Possible value are:
  - An IP address in dot notation. A port number can also be specified as part of the address, for example, 10.242.212.216:1718.
  - IP_REG_MULTICAST_DEFAULT_ADDR

time_to_live
: The time to live value in seconds. The number of seconds for which a registration is considered to be valid when repetitive registration is selected.

max_hops
: The multicast time to live value in hops. The maximum number of hops (connections between routers) that a packet can take before being discarded or returned when using multicasting.

  This field applies only to H.323 applications using gatekeeper discovery (H.225 RAS) via the default multicast registration address.

IP_VIRTBOARD

```
typedef struct
{
    unsigned short    version;
    unsigned int      total_max_calls;
    unsigned int      h323_max_calls;
    unsigned int      sip_max_calls;
    IPADDR            localIP;
    unsigned short    h323_signaling_port;
    unsigned short    sip_signaling_port;
    void              *reserved;
    unsigned short    size;
    unsigned int      sip_msginfo_mask;
    unsigned int      sup_serv_mask;
    unsigned int      h323_msginfo_mask;
    unsigned short    terminal_type
}IP_VIRTBOARD;
```

■ Description

The IP_VIRTBOARD data structure is used to store information about an IPT board device. The structure is initialized to default values by the **INIT_IP_VIRTBOARD( )** initialization function, which must be called before calling **gc_Start( )**. Default values can be overridden before calling **gc_Start( )**.

■ Field Descriptions

The fields of the IP_VIRTBOARD data structure are described as follows:

version
: The version of the structure. The correct version number is populated by the **INIT_IP_VIRTBOARD( )** function and does not need to be overriden.

total_max_calls
: The maximum total number of IPT devices that can be open concurrently. Possible values are in the range 1 to 2016 (IP_CFG_MAX_AVAILABLE_CALLS). Each IPT device can support both the H.323 and SIP protocols. The default value is 120.

h323_max_calls
: The maximum number of IPT devices that can be used for H.323 calls. Possible values are in the range 1 to 2016 (IP_CFG_MAX_AVAILABLE_CALLS). The default value is 120.

sip_max_calls
: The maximum number of IPT devices that can be used for SIP calls. Possible values are in the range 1 to 2016 (IP_CFG_MAX_AVAILABLE_CALLS). The default value is 120.

localIP
: The local IP address of type IPADDR. See IPADDR, on page 245.

h323_signaling_port
: The H.323 call signaling port. Possible values are a valid port number or IP_CFG_DEFAULT. The default H.323 signaling port is 1720.

sip_signaling_port
: The SIP call signaling port. Possible values are a valid port number or IP_CFG_DEFAULT. The default SIP signaling port is 5060.

reserved
For library use only

size
For library use only

sip_msginfo_mask (version ≥ 0x101 only)
Enables and disables access to SIP message information. Access is disabled by default. Use the value IP_SIP_MSGINFO_ENABLE to enable access to supported SIP message information fields.

sup_serv_mask (version ≥ 0x102 only)
Enables and disables the call transfer supplementary service when using the H.450.2 protocol. Access is disabled by default. Use the value IP_SUP_SERV_CALL_XFER to enable call transfer.

h323_msginfo_mask (version ≥ 0x103 only)
Enables and disables access to H.323 message information fields. Access is disabled by default. Use the value IP_H323_MSGINFO_ENABLE to enable access.

terminal_type (version ≥ 0x104 only)
Sets the Terminal Type for the virtual board which will be used during RAS registration (H.323 terminal type) and during Master Slave determination (H.245 terminal type). The value may only be changed from the default that is set by the **INIT_IP_VIRTBOARD( )** initialization function before calling **gc_Start( )**. Unsigned shorts from 0 to 255 are valid values, but the specific values 0 and 255 are reserved and will result in the terminal type being set to the default. Values larger than 255 will be truncated to 8 bits. The following symbolic values are defined:

- IP_TT_GATEWAY (Default) – Value = 60, for operation as terminal type Gateway
- IP_TT_TERMINAL – value = 50, for operation as terminal type Terminal

IPADDR

```
typedef struct
{
    unsigned char     ip_ver;
    union{
        unsigned int     ipv4;
        unsigned int     ipv6[4]
        }u_ipaddr;
}IPADDR, *PIPADDR;
```

■ Description

The IPADDR structure is used to specify a local IP address.

■ Field Descriptions

The fields of the IPADDR data structure are described as follows:

ip_ver
: The version of the local IP address. Possible values are:
  - IPVER4
  - IPVER6

u_ipaddr
: A union that contains the actual address. The datatype is different depending on whether the address is an IPv4 or an IPv6 address.

  For an IPv4 address, the address must be stored in memory using the network byte order (big endian) rather than the little-endian byte order of the Intel architecture. A socket API, **htonl( )**, is available to convert from host byte order to network byte order. As an example, to specify an IP address of 127.10.20.30, you may use either of the following C statements:

  `ipv4 = 0x1e140a7f` -or-

  `ipv4 = htonl(0x7f0a141e)`

  For more information on the byte order of IPv4 addresses, see RFC 791 and RFC 792.

IPCCLIB_START_DATA

```
typedef struct
{
    unsigned short    version;
    unsigned char     delimiter;
    unsigned char     num_boards;
    IP_VIRTBOARD      *board_list;
} IPCCLIB_START_DATA;
```

■ Description

The IPCCLIB_START_DATA structure is used to configure the IP H.323/SIP call control library when starting Global Call. Use the **INIT_IPCCLIB_START_DATA( )** function to populate a IPCCLIB_START_DATA structure with default values, then override the default values as desired.

■ Field Descriptions

The fields of the IPCCLIB_START_DATA data structure are described as follows:

version
: The version of the start structure. The correct version number is populated by the **INIT_IPCCLIB_START_DATA( )** function and does not need to be overriden.

delimiter
: An ANSI character that specifies the address string delimiter; the default delimiter is the comma (,). The specified delimiter character is used to separate the components of the destination information when using **gc_MakeCall( )**, for example.

num_boards
: The number of IPT board devices. See Section 2.3.2, "IPT Board Devices", on page 35 for more information on IPT board devices. The maximum value is 8, and the default is 2.

board_list
: A pointer to an array of IP_VIRTBOARD structures, one structure for each IPT board device. See IP_VIRTBOARD, on page 243 for more information.

IP-Specific Event Cause Codes 10

This chapter lists the IP-specific error and event cause codes and provides a description of each code. The 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.

When using **gc_DropCall( )** with H.323, only event cause codes prefixed by IPEC_H2250 or IPEC_Q931 should be specified in the **cause** parameter.

When using **gc_DropCall( )** with SIP, if the application wants to reject a call during call establishment, the relevant cause value for the **gc_DropCall( )** function can be either one of the generic Global Call cause values for dropping a call (see the **gc_DropCall( )** function description in the *Global Call API Library Reference*), or one of the cause codes prefixed by IPEC_SIP in this chapter. If the application wants to drop a call that is already connected (simply hanging up normally) the same rules apply, but the cause is not relevant in the BYE message.

When using **gc_Extension( )** to reject an incoming request to switch from audio to T.38 fax or vice versa, use only the cause codes prefixed by "IPEC_Q931Cause" for H.323, or the cause codes prefixed by "IPEC_SIPReason" for SIP.

10.1 IP-Specific Error Codes

The following IP-specific error codes are supported:

IPERR_ADDRESS_IN_USE
: The address specified is already in use. For IP networks, this will usually occur if an attempt is made to open a socket with a port that is already in use.

IPERR_ADDRESS_RESOLUTION
: Unable to resolve address to a valid IP address.

IPERR_BAD_PARAM
: Call failed because of a bad parameter.

IPERR_CALLER_ID
: Unable to allocate or copy caller ID string.

IPERR_CANT_CLOSE_CHANNEL
: As a result of the circumstances under which this channel was opened, it cannot be closed. This could occur for some protocols in the scenario when channels are opened before the call is connected. In this case, the channels should be closed and deleted after hang-up.

IPERR_CHANNEL_ACTIVE
: Media channel is already active.

IPERR_COPYING_OCTET_STRING
: Unable to copy octet string.

IPERR_COPYING_OR_RESOLVING_ALIAS
An error occurred while copying the alias. The error could be the result of a memory allocation failure or it could be an invalid alias format.

IPERR_DESTINATION_UNKNOWN
Failure to locate the host with the address given.

IPERR_DIAL_ADDR_MUST_BE_ALIAS
The address being dialed in this case may not be an IP address or domain name. It must be an alias because two intermediate addresses have already been specified, that is, Local Proxy, Remote Proxy and Gateway Address.

IPERR_DLL_LOAD_FAILED
Dynamic load of a DLL failed.

IPERR_DTMF_PENDING
Already in a DTMF generate state.

IPERR_DUP_CONF_ID
A conference ID was specified that matches an existing conference ID for another conference.

IPERR_FRAMESPERPACKET_NOT_SUPP
Setting frames-per-packet is not supported on the specified audio capability.

IPERR_GC_INVLINEDEV
Invalid line device.

IPERR_HOST_NOT_FOUND
Could not reach the party with the given host address.

IPERR_INCOMING_CALL_HANDLE
The handle passed as the incoming call handle does not refer to a valid incoming call.

IPERR_INTERNAL
An internal error occurred.

IPERR_INVALID_ADDRESS_TYPE
The address type specified did not map to any known address type.

IPERR_INVALID_CAPS
Channel open or response failed due to invalid capabilities.

IPERR_INVALID_DEST_ADDRESS
The destination address did not conform to the type specified.

IPERR_INVALID_DOMAIN_NAME
The domain name given is invalid.

IPERR_INVALID_DTMF_CHAR
Invalid DTMF character sent.

IPERR_INVALID_EMAIL_ADDRESS
The e-mail address given is invalid.

IPERR_INVALID_HOST_NAME
The host name given is invalid.

IPERR_INVALID_ID
An invalid ID was specified.

IPERR_INVALID_IP_ADDRESS
The IP address given is invalid.

IPERR_INVALID_MEDIA_HANDLE
The specified media handle is different from the already attached media handle.

IPERR_INVALID_PHONE_NUMBER
The phone number given is invalid.

IPERR_INVALID_PROPERTY
The property ID is invalid.

IPERR_INVALID_STATE
Invalid state to make this call.

IPERR_INVALID_URL_ADDRESS
The URL address given is invalid.

IPERR_INVDEVNAME
Invalid device name.

IPERR_IP_ADDRESS_NOT_AVAILABLE
The network socket layer reports that the IP address is not available. This can happen if the system does not have a correctly configured IP address.

IPERR_LOCAL_INTERNAL_PROXY_ADDR
Local internal proxy specified could not be resolved to a valid IP address or domain name.

IPERR_MEDIA_NOT_ATTACHED
No media resource was attached to the specified linedevice.

IPERR_MEMORY
Memory allocation failure.

IPERR_MULTIPLE_CAPS
Attaching a channel with multiple capabilities is not supported by this stack or it is not supported in this mode.

IPERR_MULTIPLE_DATATYPES
Attaching a channel with multiple data types (such as audio and video) is not permitted. All media types proposed for a single channel must be of the same type.

IPERR_NO_AVAILABLE_PROPOSALS
No available proposals to respond to.

IPERR_NO_CAPABILITIES_SPECIFIED
No capabilities have been specified yet. They must either be pre-configured in the configuration file or they must be set using an extended capability API.

IPERR_NO_DTMF_CAPABILITY
The remote endpoint does not have DTMF capability.

IPERR_NO_INTERSECTING_CAPABILITIES
No intersecting capability found.

IPERR_NOANSWER
Timeout due to no answer from peer.

IPERR_NOT_IMPLEMENTED
The function or property call has not been implemented. This differs from IPERR_UNSUPPORTED in that there is the implication that this is an early release which intends to implement the feature or function.

IPERR_NOT_MULTIPOINT_CAPABLE
The call cannot be accepted into a multipoint conference because there is no known multipoint controller, or the peer in a point-to-point conference is not multipoint capable.

IPERR_NULL_ADDRESS
Address given is NULL.

IPERR_NULL_ALIAS
The alias specified is NULL or empty.

IPERR_OK
Successful completion.

IPERR_PEER_REJECT
Peer has rejected the call placed from this endpoint.

IPERR_PENDING_RENEGOTIATION
A batched channel renegotiation is already pending. This implementation does not support queuing of batched renegotiation.

IPERR_PROXY_GATEWAY_ADDR
Two intermediate addresses were already specified in the local internal proxy and remote proxy addresses. The gateway address in this case cannot be used.

IPERR_REMOTE_PROXY_ADDR
Remote proxy specified could not be resolved to a valid IP address or domain name.

IPERR_SERVER_REGISTRATION_FAILED
Attempt to register with the registration and admission server (RAS) failed.

IPERR_STILL_REGISTERED
The address object being deleted is still registered and cannot be deleted until it is unregistered.

IPERR_TIMEOUT
Timeout occurred while executing an internal function.

IPERR_UNAVAILABLE
The requested data is unavailable.

IPERR_UNDELETED_OBJECTS
The object being deleted has child objects that have not been deleted.

IPERR_UNICODE_TO_ASCII
Unable to convert the string or character from unicode or wide character format to ASCII.

IPERR_UNINITIALIZED
The stack has not been initialized.

IPERR_UNKNOWN_API_GUID
This is the result of either passing in a bogus GUID or one that is not found in the current DLL or executable.

IPERR_UNRESOLVABLE_DEST_ADDRESS
: No Gateway, Gatekeeper, or Proxy is specified, therefore the destination address must be a valid resolvable address. In the case of IP based call control, the address specified should be an IP address or a resolvable host or domain name.

IPERR_UNRESOLVABLE_HOST_NAME)
: The host or domain name could not be resolved to a valid address. This will usually occur if the host or domain name is not valid or is not accessible over the existing network.

IPERR_UNSUPPORTED
: This function or property call is unsupported in this configuration or implementation of stack. This differs from IPERR_NOT_IMPLEMENTED in that it implies no future plan to support this feature of property.

10.2 Error Codes When Using H.323

The following error codes are supported:

IPEC_addrRegistrationFailed
: Registration with the Registration and Admission server failed.

IPEC_addrListenFailed
: Stack was unable to register to listen for incoming calls.

IPEC_CHAN_REJECT_unspecified
: No cause for rejection specified.

IPEC_CHAN_REJECT_dataTypeNotSupported
: The terminal was not capable of supporting the dataType indicated in OpenLogicalChannel.

IPEC_CHAN_REJECT_dataTypeNotAvailable
: The terminal was not capable of supporting the dataType indicated in OpenLogicalChannel simultaneously with the dataTypes of logical channels that are already open.

IPEC_CHAN_REJECT_unknownDataType
: The terminal did not understand the dataType indicated in OpenLogicalChannel.

IPEC_CHAN_REJECT_insuffientBandwdith
: The channel could not be opened because permission to use the requested bandwidth for the logical channel was denied.

IPEC_CHAN_REJECT_unsuitableReverseParameters
: This code shall only be used to reject a bi-directional logical channel request when the only reason for rejection is that the requested parameters are inappropriate.

IPEC_CHAN_REJECT_dataTypeALCombinationNotSupported
: The terminal was not capable of supporting the dataType indicated in OpenLogicalChannel simultaneously with the Adaptation Layer type indicated in H223LogicalChannelParameters.

IPEC_CHAN_REJECT_multicastChannelNotAllowed
: Multicast Channel could not be opened.

IPEC_CHAN_REJECT_separateStackEstablishmentFailed
: A request to run the data portion of a call on a separate stack failed.

IPEC_CHAN_REJECT_invalidSessionID
: Attempt by the slave to set the SessionID when opening a logical channel to the master.

IPEC_CHAN_REJECT_masterSlaveConflict
: Attempt by the slave to open logical channel in which the master has determined a conflict may occur.

IPEC_CHAN_REJECT_waitForCommunicationMode
: Attempt to open a logical channel before the MC has transmitted the CommunicationModeCommand.

IPEC_CHAN_REJECT_invalidDependentChannel
: Attempt to open a logical channel with a dependent channel specified that is not present.

IPEC_CHAN_REJECT_replacementForRejected
: A logical channel of the type attempted cannot be opened using the replacement **For** parameter.The transmitter may wish to re-try by first closing the logical channel that is to be replaced, and then opening the replacement.

IPEC_CALL_END_timeout
: A callback was received because a local timer expired.

IPEC_InternalError
: An internal error occurred while executing asynchronously.

IPEC_INFO_NONE_NOMORE
: No more digits are available.

IPEC_INFO_PRESENT_MORE
: The requested digits are now available. More/additional digits are available.

IPEC_INFO_PRESENT_ALL
: The requested digits are now available.

IPEC_INFO_NONE_TIMEOUT
: No digits are available; timed out.

IPEC_INFO_SOME_NOMORE
: Only some digits are available, no more digits will be received.

IPEC_INFO_SOME_TIMEOUT
: Only some digits are available; timed out.

IPEC_NO_MATCHING_CAPABILITIES
: No intersection was found between the proposed and matching capabilities.

IPEC_REG_FAIL_duplicateAlias
: The alias used to register with the Registration and Admission server is already registered. This failure typically results if the endpoint is already registered. It could also occur with some servers if a registration is attempted too soon after unregistering using the same alias.

IPEC_REG_FAIL_invalidCallSigAddress
: Server registration failed due to an invalid call signalling address specified.

IPEC_REG_FAIL_invalidAddress
: The local host address specified for communicating with the server is invalid.

IPEC_REG_FAIL_invalidAlias
: The alias specified did not conform to the format rules for the type of alias specified.

IPEC_REG_FAIL_invalidTermType
An invalid terminal type was specified with the registration request.

IPEC_REG_FAIL_invalidTransport
The transport type of the local host's address is not supported by the server.

IPEC_REG_FAIL_qosNotSupported
The registration request announced a transport QoS that was not supported by the server.

IPEC_REG_FAIL_reRegistrationRequired
Registration permission has expired. Registration should be performed again.

IPEC_REG_FAIL_resourcesUnavailable
The server rejected the registration request due to unavailability of resources. This typically occurs if the server has already reached the maximum number of registrations it was configured to accept.

IPEC_REG_FAIL_securityDenied
The server denied access for security reasons. This can occur if the password supplied does not match the password on file for the alias being registered.

IPEC_REG_FAIL_unknown
The server refused to allow registration for an unknown reason.

IPEC_REG_FAIL_serverDown
The server has gone down or is no longer responding.

IPEC_MEDIA_startSessionFailed
Attempt to call **gc_media_StartSession( )** (an internal function) after establishing media channel returned error.

IPEC_MEDIA_TxFailed
Attempt to establish or terminate a Tx channel with attached capabilities failed. The application is expected to keep the Rx capabilities unchanged in the next call to **gc_AttachEx( )**.

IPEC_MEDIA_RxFailed
Attempt to establish or terminate an Rx channel with attached capabilities failed. The application is expected to keep the Tx capabilities unchanged in the next call to **gc_AttachEx( )**.

IPEC_MEDIA_TxRxFailed
Attempts to establish or terminate Tx and Rx channels with attached capabilities failed.

IPEC_MEDIA_OnlyTxFailed
Attempts to establish a Tx channel with attached capabilities failed. The status of other media channel is unavailable. Relevant to the GCEV_MEDIA_REJ event.

IPEC_MEDIA_OnlyRxFailed
Attempts to establish an Rx channel with attached capabilities failed. The status of other media channel is unavailable. Relevant to the GCEV_MEDIA_REJ event.

IPEC_MEDIA_TxRequired
Attempts to establish a Tx channel with attached capabilities failed.

IPEC_MEDIA_RxRequired
Attempts to establish an Rx channel with attached capabilities failed.

IPEC_TxRx_Fail
Both channels have failed to open.

IPEC_Tx_FailTimeout
A Tx channel failed to open because of timeout.

IPEC_Rx_FailTimeout
An Rx channel failed to open because of timeout.

IPEC_Tx_Fail
A Tx channel failed to open for an unknown reason.

IPEC_Rx_Fail
An Rx channel failed to open for an unknown reason.

IPEC_TxRx_FailTimeout
Both the Tx and Rx channels failed because of a timeout.

IPEC_TxRx_Rej
Both the Tx and Rx channels were rejected for an unknown reason.

IPEC_Tx_Rej
Opening of a Tx channel was rejected for unknown reasons.

IPEC_Rx_Rej
Opening of an Rx channel was rejected for unknown reasons.

IPEC_CHAN_FAILURE_unspecified
The channel failed to open/close because of an unspecified reason.

IPEC_CHAN_FAILURE_timeout
The channel failed to open/close because of a timeout.

IPEC_CHAN_FAILURE_localResources
The channel failed to open/close because of limited resources.

IPEC_FAIL_TxRx_unspecified
Both the Tx and Rx channels failed to open for unspecified reasons.

IPEC_FAIL_TxUnspecifiedRxTimeout
A Tx channel failed to open for unspecified reasons and the Rx channel failed to open because of a timeout.

IPEC_FAILTxUnspecifiedRxResourceUnsuff
A Tx channel failed to open for unspecified reasons and the Rx channel failed to open because of insufficient resources.

IPEC_FAIL_RxUnspecifiedTxTimeout
An Rx channel failed to open for unspecified reasons and the Tx channel failed to open because of a timeout.

IPEC_FAIL_RXUnspecifiedTxResourceUnsuff
An Rx channel failed to open for unspecified reasons and the Tx channel failed to open because of insufficient resources.

IPEC_FAIL_TxTimeoutRxUnspecified
A Tx channel failed to open because of a timeout and the Rx channel failed to open for unspecified reasons.

IPEC_FAIL_TxRxTimeout
The Tx and Rx channels both failed to open because of a timeout.

IPEC_FAIL_TxTimeoutRxResourceUnsuff
A Tx channel failed to open because of a timeout and the Rx channel failed to open because of insufficient resources.

IPEC_FAIL_RxTimeoutTXUnspecified
An Rx channel failed because of a timeout and the Tx channel failed for unspecified reasons.

IPEC_FAIL_RxTimeoutTxResourceUnsuff
A Tx channel failed to open because of a timeout and the Rx channel failed to open because of insufficient resources.

IPEC_FAIL_TxResourceUnsuffRxUnspecified
A Tx channel failed to open because of insufficient resources and the Rx channel failed to open for unspecified reasons.

IPEC_FAIL_TxResourceUnsuffRxTimeout
A Tx channel failed to open because of insufficient resources and the Rx channel failed to open because of a timeout.

IPEC_FAIL_TxRxResourceUnsuff
Tx and Rx channels failed to open because of insufficient resources.

IPEC_FAIL_RxResourceUnsuffTxUnspecified
A Tx channel failed to open for unspecified reasons and the Rx channel failed to open because of insufficient resources.

IPEC_FAIL_RxResourceUnsuffTxTimeout
A Tx channel failed to open because of a timeout and the Rx channel failed to open because of insufficient resources.

10.3 Internal Disconnect Reasons

The following internal disconnect reasons are supported when using H.323:

IPEC_InternalReasonBusy (0x3e9, 1001 decimal)
Cause 01; Busy

IPEC_InternalReasonCallCompletion (0x3ea, 1002 decimal)
Cause 02; Call Completion

IPEC_InternalReasonCanceled (0x3eb, 1003 decimal)
Cause 03; Cancelled

IPEC_InternalReasonCongestion (0x3ec, 1004 decimal)
Cause 04; Network congestion

IPEC_InternalReasonDestBusy (0x3ed, 1005 decimal)
Cause 05; Destination busy

IPEC_InternalReasonDestAddrBad (0x3ee, 1006 decimal)
Cause 06; Invalid destination address

IPEC_InternalReasonDestOutOfOrder (0x3ef, 1007 decimal)
Cause 07; Destination out of order

IPEC_InternalReasonDestUnobtainable (0x3f0, 1008 decimal)
Cause 08; Destination unobtainable

IPEC_InternalReasonForward (0x3f1, 1009 decimal)
Cause 09; Forward

IPEC_InternalReasonIncompatible (0x3f2, 1010 decimal)
Cause 10; Incompatible

IPEC_InternalReasonIncomingCall, (0x3f3, 1011 decimal)
Cause 11; Incoming call

IPEC_InternalReasonNewCall (0x3f4, 1012 decimal)
Cause 12; New call

IPEC_InternalReasonNoAnswer (0x3f5, 1013 decimal)
Cause 13; No answer from user

IPEC_InternalReasonNormal (0x3f6, 1014 decimal)
Cause 14; Normal clearing

IPEC_InternalReasonNetworkAlarm (0x3f7, 1015 decimal)
Cause 15; Network alarm

IPEC_InternalReasonPickUp (0x3f8, 1016 decimal)
Cause 16; Pickup

IPEC_InternalReasonProtocolError (0x3f9, 1017 decimal)
Cause 17; Protocol error

IPEC_InternalReasonRedirection (0x3fa, 1018 decimal)
Cause 18; Redirection

IPEC_InternalReasonRemoteTermination (0x3fb, 1019 decimal)
Cause 19; Remote termination

IPEC_InternalReasonRejection (0x3fc, 1020 decimal)
Cause 20; Call rejected

IPEC_InternalReasonSIT (0x3fd, 1021 decimal)
Cause 21; Special Information Tone (SIT)

IPEC_InternalReasonSITCustIrreg (0x3fe, 1022 decimal)
Cause 22; SIT, Custom Irregular

IPEC_InternalReasonSITNoCircuit (0x3ff, 1023 decimal)
Cause 23; SIT, No Circuit

IPEC_InternalReasonSITReorder (0x400, 1024 decimal)
Cause 24; SIT, Reorder

IPEC_InternalReasonTransfer (0x401, 1025 decimal)
Cause 25; Transfer

IPEC_InternalReasonUnavailable (0x402, 1026 decimal)
Cause 26; Unavailable

IPEC_InternalReasonUnknown (0x403, 1027 decimal)
Cause 27; Unknown cause

IPEC_InternalReasonUnallocatedNumber (0x404, 1028 decimal)
Cause 28; Unallocated number

IPEC_InternalReasonNoRoute (0x405, 1029 decimal)
Cause 29; No route

IPEC_InternalReasonNumberChanged (0x406, 1030 decimal)
Cause 30; Number changed

IPEC_InternalReasonOutOfOrder (0x407, 1031 decimal)
Cause 31; Destination out of order

IPEC_InternalReasonInvalidFormat (0x408, 1032 decimal)
Cause 32; Invalid format

IPEC_InternalReasonChanUnavailable (0x409, 1033 decimal)
Cause 33; Channel unavailable

IPEC_InternalReasonChanUnacceptable (0x40a, 1034 decimal)
Cause 34; Channel unacceptable

IPEC_InternalReasonChanNotImplemented (0x40b, 1035 decimal)
Cause 35; Channel not implemented

IPEC_InternalReasonNoChan (0x40c, 1036 decimal)
Cause 36; No channel

IPEC_InternalReasonNoResponse (0x40d, 1037 decimal)
Cause 37; No response

IPEC_InternalReasonFacilityNotSubscribed (0x40e, 1038 decimal)
Cause 38; Facility not subscribed

IPEC_InternalReasonFacilityNotImplemented (0x40f, 1039 decimal)
Cause 39; Facility not implemented

IPEC_InternalReasonServiceNotImplemented (0x410, 1040 decimal)
Cause 40; Service not implemented

IPEC_InternalReasonBarredInbound (0x411, 1041 decimal)
Cause 41; Barred inbound calls

IPEC_InternalReasonBarredOutbound (0x412, 1042 decimal)
Cause 42; Barred outbound calls

IPEC_InternalReasonDestIncompatible (0x413, 1043 decimal)
Cause 43; Destination incompatible

IPEC_InternalReasonBearerCapUnavailable (0x414, 1044 decimal)
Cause 44; Bearer capability unavailable

10.4 Event Cause Codes and Failure Reasons When Using H.323

The following event cause codes apply when using H.323.

H.225.0 Cause Codes

IPEC_H2250ReasonNoBandwidth (0x7d0, 2000 decimal)
: Maps to Q.931/Q.850 cause 34 - No circuit or channel available; indicates that there is no appropriate circuit/channel presently available to handle the call.

IPEC_H2250ReasonGatekeeperResource (0x7d1, 2001 decimal)
: Maps to Q.931/Q.850 cause 47 - Resource unavailable; used to report a resource unavailable event only when no other cause in the resource unavailable class applies.

IPEC_H2250ReasonUnreachableDestination (0x7d2, 2002 decimal)
: Maps to Q.931/Q.850 cause 3 - No route to destination; indicates that the called party cannot be reached because the network through which the call has been routed does not serve the destination desired.

IPEC_H2250ReasonDestinationRejection (0x7d3, 2003 decimal)
: Maps to Q.931/Q.850 cause 16 - Normal call clearing - indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared.

IPEC_H2250ReasonInvalidRevision (0x7d4, 2004 decimal)
: Maps to Q.931/Q.850 cause 88 - Incompatible destination; indicates that the equipment sending this cause has received a request to establish a call which has low layer compatibility, high layer compatibility, or other compatibility attributes (for example, data rate) which cannot be accommodated.

IPEC_H2250ReasonNoPermission (0x7d5, 2005 decimal)
: Maps to Q.931/Q.850 cause 111 - Interworking, unspecified.

IPEC_H2250ReasonUnreachableGatekeeper (0x7d6, 2006 decimal)
: Maps to Q.931/Q.850 cause 38 - Network out of order; indicates that the network is not functioning correctly and that the condition is likely to last a relatively long period of time, for example, immediately re-attempting the call is not likely to be successful.

IPEC_H2250ReasonGatewayResource (0x7d7, 2007 decimal)
: Maps to Q.931/Q.850 cause 42 - Switching equipment congestion; indicates that the switching equipment generating this cause is experiencing a period of high traffic.

IPEC_H2250ReasonBadFormatAddress (0x7d8, 2008 decimal)
: Maps to Q.931/Q.850 cause 28 - Invalid number format; indicates that the called party cannot be reached because the called party number is not in a valid format or is incomplete.

IPEC_H2250ReasonAdaptiveBusy (0x7d9, 2009 decimal)
: Maps to Q.931/Q.850 cause 41 - Temporary failure; indicates that the network is not functioning correctly and that the condition is not likely to last for a long period of time, for example, the user may wish to try another call attempt almost immediately.

IPEC_H2250ReasonInConf (0x7da, 2010 decimal)
Maps to Q.931/Q.850 cause 17 - User busy; used to indicate that the called party is unable to accept another call because the user busy condition has been encountered. This cause value may be generated by the called user or by the network.

IPEC_H2250ReasonUndefinedReason (0x7db, 2011 decimal)
Maps to Q.931/Q.850 cause 31 - Normal, unspecified; Normal, unspecified; used to report a normal event only when no other cause in the normal class applies.

IPEC_H2250ReasonFacilityCallDeflection (0x7dc, 2012 decimal)
Maps to Q.931/Q.850 cause 16 - Normal call clearing - indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared.

IPEC_H2250ReasonSecurityDenied (0x7dd, 2013 decimal)
Maps to Q.931/Q.850 cause 31 - Normal, unspecified; Normal, unspecified; used to report a normal event only when no other cause in the normal class applies.

IPEC_H2250ReasonCalledPartyNotRegistered (0x7de, 2014 decimal)
Maps to Q.931/Q.850 cause 20 - Subscriber absent; used when a mobile station has logged off, radio contact is not obtained with a mobile station or if a personal telecommunication user is temporarily not addressable at any user-network interface.

IPEC_H2250ReasonCallerNotRegistered (0x7df, 2015 decimal)
Maps to Q.931/Q.850 cause 31 - Normal, unspecified; used to report a normal event only when no other cause in the normal class applies.

Q.931 Cause Codes

IPEC_Q931Cause01UnassignedNumber (0xbb9, 3001 decimal)
Q.931 cause 01 - Unallocated (unassigned) number; indicates that the called party cannot be reached because. Although the called party number is in a valid format, it is not currently allocated (assigned).

IPEC_Q931Cause02NoRouteToSpecifiedTransitNetwork (0xbba, 3002 decimal)
Q.931 cause 02 - No route to specified transit network (national use); indicates that the equipment sending this cause has received a request to route the call through a particular transit network which it does not recognize. The equipment sending this cause does not recognize the transit network either because the transit network does not exist or because that particular transit network, while it does exist, does not serve the equipment which is sending this cause. This cause is supported on a network-dependent basis.

IPEC_Q931Cause03NoRouteToDestination (0xbbb, 3003 decimal)
Q.931 cause 03 - No route to destination; indicates that the called party cannot be reached because the network through which the call has been routed does not serve the destination desired. This cause is supported on a network-dependent basis.

IPEC_Q931Cause06ChannelUnacceptable (0xbbe, 3006 decimal)
Q.931 cause 06 - Channel unacceptable; indicates that the channel most recently identified is not acceptable to the sending entity for use in this call.

IPEC_Q931Cause07CallAwardedAndBeingDeliveredInAnEstablishedChannel (0xbbf, 3007 decimal)

Q.931 cause 07 - Call awarded and being delivered in an established channel; indicates that the user has been awarded the incoming call, and that the incoming call is being connected to a channel already established to that user for similar calls (e.g. packet-mode X.25 virtual calls).

IPEC_Q931Cause16NormalCallClearing (0xbc8, 3016 decimal)

Q.931 cause 16 - Normal call clearing; indicates that the call is being cleared because one of the user's involved in the call has requested that the call be cleared. Under normal situations, the source of this cause is not the network.

IPEC_Q931Cause17UserBusy (0xbc9, 3017 decimal)

Q.931 cause 17 - User busy; used to indicate that the called party is unable to accept another call because the user busy condition has been encountered. This cause value may be generated by the called user or by the network.

IPEC_Q931Cause18NoUserResponding (0xbca, 3018 decimal)

Q.931 cause 18 - No user responding; used when a called party does not respond to a call establishment message with either an alerting or connect indication within the prescribed period of time allocated.

IPEC_Q931Cause19UserAlertingNoAnswer (0xbcb, 3019 decimal)

Q.931 cause 19 - No answer from user (user alerted); used when the called party has been alerted but does not respond with a connect indication within a prescribed period of time. This cause is not necessarily generated by Q.931 procedures but may be generated by internal network timers.

IPEC_Q931Cause21CallRejected (0xbcd, 3021 decimal)

Q.931 cause 21 - Call rejected; indicates that the equipment sending this cause does not wish to accept this call, although it could have accepted the call because the equipment sending this cause is neither busy nor incompatible. This cause may also be generated by the network, indicating that the call was cleared due to a supplementary service constraint. The diagnostic field may contain additional information about the supplementary service and reason for rejection.

IPEC_Q931Cause22NumberChanged (0xbce, 3022 decimal)

Q.931 cause 22 - Number changed; returned to a calling party when the called party number indicated by the calling party is no longer assigned. The new called party number may optionally be included in the diagnostic field. If a network does not support this cause value, cause No. 1, unallocated (unassigned) number should be used.

IPEC_Q931Cause26NonSelectUserClearing (0xbd2, 3026 decimal)

Q.931 cause 26 - Non-selected user clearing; indicates that the user has not been awarded the incoming call.

IPEC_Q931Cause27DestinationOutOfOrder (0xbd3, 3027 decimal)

Q.931 cause 27 - Destination out of order; indicates that the destination indicated by the user cannot be reached because the interface to the destination is not functioning correctly. The term "not functioning correctly" indicates that a signalling message was unable to be delivered to the remote party, for example, a physical layer or data link layer failure at the remote party, or user equipment off-line.

IPEC_Q931Cause28InvalidNumberFormatIncompleteNumber (0xbd4, 3028 decimal)
Q.931 cause 28 - Invalid number format (address incomplete); indicates that the called party cannot be reached because the called party number is not in a valid format or is not complete. Note: This condition may be determined immediately after reception of an ST signal or on time-out after the last received digit.

IPEC_Q931Cause29FacilityRejected (0xbd5, 3029 decimal)
Q.931 cause 29 - Facility rejected; returned when a supplementary service requested by the user cannot be provided by the network.

IPEC_Q931Cause30ResponseToSTATUSENQUIRY (0xbd6, 3030 decimal)
Q.931 cause 30 - Response to STATUS ENQUIRY; included in the STATUS message when the reason for generating the STATUS message was the prior receipt of a STATUS ENQUIRY message.

IPEC_Q931Cause31NormalUnspecified (0xbd7, 3031 decimal)
Q.931 cause 31 - Normal, unspecified; used to report a normal event only when no other cause in the normal class applies.

IPEC_Q931Cause34NoCircuitChannelAvailable (0xbda, 3034 decimal)
Q.931 cause 34 - No circuit/channel available; indicates that there is no appropriate circuit/channel presently available to handle the call.

IPEC_Q931Cause38NetworkOutOfOrder (0xbde, 3038 decimal)
Q.931 cause 38 - Network out of order; indicates that the network is not functioning correctly and that the condition is likely to last a relatively long period of time, that is, immediately re-attempting the call is not likely to be successful.

IPEC_Q931Cause41TemporaryFailure (0xbe1, 3041 decimal)
Q.931 cause 41 - Temporary failure; indicates that the network is not functioning correctly and that the condition is not likely to last a long period of time, that is, the user may wish to try another call attempt almost immediately.

IPEC_Q931Cause42SwitchingEquipmentCongestion (0xbe2, 3042 decimal)
Q.931 cause 42 - Switching equipment congestion; indicates that the switching equipment generating this cause is experiencing a period of high traffic.

IPEC_Q931Cause43AccessInformationDiscarded (0xbe3, 3043 decimal)
Q.931 cause 43 - Access information discarded; indicates that the network could not deliver access information to the remote user as requested, that is, user-to-user information, low layer compatibility, high layer compatibility, or sub-address, as indicated in the diagnostic. The particular type of access information discarded is optionally included in the diagnostic.

IPEC_Q931Cause44RequestedCircuitChannelNotAvailable (0xbe4, 3044 decimal)
Q.931 cause 44 - Requested circuit/channel not available; returned when the circuit or channel indicated by the requesting entity cannot be provided by the other side of the interface.

IPEC_Q931Cause47ResourceUnavailableUnspecified (0xbe7, 3047 decimal)
Q.931 cause 47 - Resource unavailable, unspecified; used to report a resource unavailable event only when no other cause in the resource unavailable class applies.

IPEC_Q931Cause57BearerCapabilityNotAuthorized (0xbf1, 3057 decimal)
Q.931 cause 57 - Bearer capability not authorized; indicates that the user has requested a bearer capability that is implemented by the equipment that generated this cause but the user is not authorized to use.

IPEC_Q931Cause58BearerCapabilityNotPresentlyAvailable (0xbf2, 3058 decimal)
: Q.931 cause 58 - Bearer capability not presently available; indicates that the user has requested a bearer capability that is implemented by the equipment that generated this cause but it is not available at this time.

IPEC_Q931Cause63ServiceOrOptionNotAvailableUnspecified (0xbf7, 3063 decimal)
: Q.931 cause 63 - Service or option not available, unspecified; used to report a service or option not available event only when no other cause in the service or option not available class applies.

IPEC_Q931Cause65BearCapabilityNotImplemented (0xbf9, 3065 decimal)
: Q.931 cause 65 - Bearer capability not implemented; indicates that the equipment sending this cause does not support the bearer capability requested.

IPEC_Q931Cause66ChannelTypeNotImplemented (0xbfa, 3066 decimal)
: Q.931 cause 66 - Channel type not implemented; indicates that the equipment sending this cause does not support the channel type requested.

IPEC_Q931Cause69RequestedFacilityNotImplemented (0xbfd, 3069 decimal)
: Q.931 cause 69 - Requested facility not implemented; indicates that the equipment sending this cause does not support the requested supplementary service.

IPEC_Q931Cause70OnlyRestrictedDigitalInformationBearerCapabilityIsAvailable (0xbfe, 3070 decimal)
: Q.931 cause 70 - Only restricted digital information bearer capability is available (national use); indicates that the calling party has requested an unrestricted bearer service but that the equipment sending this cause only supports the restricted version of the requested bearer capability.

IPEC_Q931Cause79ServiceOrOptionNotImplementedUnspecified (0xc07, 3079 decimal)
: Q.931 cause 79 - Service or option not implemented, unspecified; used to report a service or option not implemented event only when no other cause in the service or option not implemented class applies.

IPEC_Q931Cause81InvalidCallReferenceValue (0xc09, 3081 decimal)
: Q.931 cause 81 - Invalid call reference value; indicates that the equipment sending this cause has received a message with a call reference that is not currently in use on the user-network interface.

IPEC_Q931Cause82IdentifiedChannelDoesNotExist (0xc0a, 3082 decimal)
: Q.931 cause 82 - Identified channel does not exist; indicates that the equipment sending this cause has received a request to use a channel not activated on the interface for a call. For example, if a user has subscribed to those channels on a primary rate interface numbered from 1 to 12 and the user equipment or the network attempts to use channels 13 through 23, this cause is generated.

IPEC_Q931Cause83AsuspendedCallExistsButThisCallIdentityDoesNot (0xc0b, 3083 decimal)
: Q.931 cause 83 - A suspended call exists, but this call identity does not; indicates that a call resume has been attempted with a call identity that differs from that in use for any presently suspended call(s).

IPEC_Q931Cause84CallIdentityInUse (0xc0c, 3084 decimal)
: Q.931 cause 84 - Call identity in use; indicates that the network has received a call suspended request containing a call identity (including the null call identity) that is already in use for a suspended call within the domain of interfaces over which the call might be resumed.

IPEC_Q931Cause85NoCallSuspended (0xc0d, 3085 decimal)
Q.931 cause 85 - No call suspended; indicates that the network has received a call resume request containing a call identity information element that presently does not indicate any suspended call within the domain of interfaces over which calls may be resumed.

IPEC_Q931Cause86CallHavingTheRequestedCallIdentityHasBeenCleared (0xc0e, 3086 decimal)
Q.931 cause 86 - Call having the requested call identity has been cleared; indicates that the network has received a call resume request containing a call identity information element indicating a suspended call that has in the meantime been cleared while suspended (either by network timeout or by the remote user).

IPEC_Q931Cause88IncompatibleDestination (0xc10, 3088 decimal)
Q.931 cause 88 - Incompatible destination; indicates that the equipment sending this cause has received a request to establish a call that has low layer compatibility, high layer compatibility, or other compatibility attributes (for example, data rate) that cannot be accommodated.

IPEC_Q931Cause91InvalidTransitNetworkSelection (0xc13, 3091 decimal)
Q.931 cause 91 - Invalid transit network selection (national use); indicates that a transit network identification was received that is of an incorrect format as defined by Annex C/Q.931.

IPEC_Q931Cause95InvalidMessageUnspecified (0xc17, 3095 decimal)
Q.931 cause 95 - Invalid message, unspecified; used to report an invalid message event only when no other cause in the invalid message class applies.

IPEC_Q931Cause96MandatoryInformationElementMissing (0xc18, 3096 decimal)
Q.931 cause 96 - Mandatory information element is missing; indicates that the equipment sending this cause has received a message that is missing an information element that must be present in the message before that message can be processed.

IPEC_Q931Cause97MessageTypeNonExistentOrNotImplemented (0xc19, 3097 decimal)
Q.931 cause 97 - Message type non-existent or not implemented; indicates that the equipment sending this cause has received a message with a message type it does not recognize either because 1) the message type is not defined or 2) the message type is defined but not implemented by the equipment sending this cause.

IPEC_Q931Cause100InvalidInformationElementContents (0xc1c, 3100 decimal)
Q.931 cause 100 - Invalid information element contents; indicates that the equipment sending this cause has received an information element that it has implemented; however, one or more fields in the information element are coded in such a way that has not been implemented by the equipment sending this cause.

IPEC_Q931Cause101MessageNotCompatibleWithCallState (0xc1d, 3101 decimal)
Q.931 cause 101 - Message not compatible with call state; indicates that a message that is incompatible with the call state has been received.

IPEC_Q931Cause102RecoveryOnTimeExpiry (0xc1e, 3102 decimal)
Q.931 cause 102 - Recovery on timer expiry; indicates that a procedure has been initiated by the expiry of a timer in association with error handling procedures.

IPEC_Q931Cause111ProtocolErrorUnspecified (0xc27, 3111 decimal)
Q.931 cause 111 - Protocol error, unspecified; used to report a protocol error event only when no other cause in the protocol error class applies.

IPEC_Q931Cause127InterworkingUnspecified (0xc37, 3127 decimal)
Q.931 cause 127 - Interworking, unspecified; indicates that there has been interworking with a network that does not provide causes for the actions it takes. Thus, the precise cause for a message that is being sent cannot be ascertained.

RAS Failure Reasons

IPEC_RASReasonResourceUnavailable (0xfa1, 4001 decimal)
Resources have been exhausted. (In GRJ, RRJ, ARJ, and LRJ messages.)

IPEC_RASReasonInsufficientResources (0xfa2, 4002 decimal)
Insufficient resources to complete the transaction. (In BRJ messages.)

IPEC_RASReasonInvalidRevision (0xfa3, 4003 decimal)
The registration version is invalid. (In GRJ, RRJ, and BRJ messages.)

IPEC_RASReasonInvalidCallSignalAddress (0xa4, 4004 decimal)
The call signal address is invalid. (In RRJ messages.)

IPEC_RASReasonInvalidIPEC_RASAddress (0xfa5, 4005 decimal)
The supplied address is invalid. (In RRJ messages.)

IPEC_RASReasonInvalidTerminalType (0xfa6, 4006 decimal)
The terminal type is invalid. (In RRJ messages.)

IPEC_RASReasonInvalidPermission (0xfa7, 4007 decimal)
Permission has expired. (In ARJ messages.)

A true permission violation. (In BRJ messages.)

Exclusion by administrator or feature. (In LRJ messages.)

IPEC_RASReasonInvalidConferenceID (0xfa8, 4008 decimal)
Possible revision. (In BRJ messages.)

IPEC_RASReasonInvalidEndpointID (0xfa9, 4009 decimal)
The endpoint registration ID is invalid. (In ARJ messages.)

IPEC_RASReasonCallerNotRegistered (0xfaa, 4010 decimal)
The call originator is not registered. (In ARJ messages.)

IPEC_RASReasonCalledPartyNotRegistered (0xfab, 4011 decimal)
Unable to translate the address. (In ARJ messages.)

IPEC_RASReasonDiscoveryRequired (0xfac, 4012 decimal)
Registration permission has expired. (In RRJ messages.)

IPEC_RASReasonDuplicateAlias (0xfad, 4013 decimal)
The alias is registered to another endpoint. (In RRJ messages.)

IPEC_RASReasonTransportNotSupported (0xfae, 4014 decimal)
One or more of the transport addresses are not supported. (In RRJ messages.)

IPEC_RASReasonCallInProgress (0xfaf, 4015 decimal)
A call is already in progress. (In URJ messages.)

IPEC_RASReasonRouteCallToGatekeeper (0xfb0, 4016 decimal)
The call has been routed to a gatekeeper. (In ARJ messages.)

IPEC_RASReasonRequestToDropOther (0xfb1, 4017 decimal)
Unable to request to drop the call for others. (In DRJ messages.)

IPEC_RASReasonNotRegistered (0xfb2, 4018 decimal)
Not registered with a gatekeeper. (In DRJ, LRJ, and INAK messages.)

IPEC_RASReasonUndefined (0xfb3, 4019 decimal)
Unknown reason. (In GRJ, RRJ, URJ, ARJ, BRJ, LRJ, and INAK messages.)

IPEC_RASReasonTerminalExcluded (0xfb4, 4020 decimal)
Permission failure and not a resource failure. (In GRQ messages.)

IPEC_RASReasonNotBound (0xfb5, 4021 decimal)
Discovery permission has expired. (In BRJ messages.)

IPEC_RASReasonNotCurrentlyRegistered (0xfb6, 4022 decimal)
The endpoint is not registered. (In URJ messages.)

IPEC_RASReasonRequestDenied (0xfb7, 4023 decimal)
No bandwidth is available. (In ARJ messages.)

Unable to find location. (In LRJ messages.)

IPEC_RASReasonLocationNotFound (0xfb8, 4024 decimal)
Unable to find location. (In LRJ messages.)

IPEC_RASReasonSecurityDenial (0xfb9, 4025 decimal)
Security access has been denied. (In GRJ, RRJ, URJ, ARJ, BRJ, LRJ, DRJ, and INAK messages.)

IPEC_RASTransportQOSNotSupported (0xfba, 4026 decimal)
QOS is not supported by this gatekeeper. (In RRJ messages.)

IPEC_RASResourceUnavailable (0xfbb, 4027 decimal)
Resources have been exhausted. (In GRJ, RRJ, ARJ and LRJ messages.)

IPEC_RASInvalidAlias (0xfbc, 4028 decimal)
The alias is not consistent with gatekeeper rules. (In RRJ messages.)

IPEC_RASPermissionDenied (0xfbd, 4029 decimal)
The requesting user is not allowed to unregistered the specified user. (In URJ messages.)

IPEC_RASQOSControlNotSupported (0xfbe, 4030 decimal)
QOS control is not supported. (In ARJ messages.)

IPEC_RASIncompleteAddress (0xfbf, 4031 decimal)
The user address is incomplete. (In ARJ messages.)

IPEC_RASFullRegistrationRequired (0xfc0, 4032 decimal)
Registration permission has expired. (In RRJ messages.)

IPEC_RASRouteCallToSCN (0xfc1, 4033 decimal)
The call was routed to a switched circuit network. (In ARJ and LRJ messages.)

IPEC_RASAliasesInconsistent (0xfc2, 4034 decimal)
Multiple aliases in the request identify separate people. (In ARJ and LRJ messages.)

10.5 Failure Response Codes When Using SIP

The following failure response codes apply when using SIP. Each code is followed by a description. The codes are listed in code value order.

Request Failure Response Codes (4xx)

IPEC_SIPReasonStatus400BadRequest (0x1518, 5400 decimal)
SIP Request Failure Response 400 - Bad Request - The request could not be understood due to malformed syntax. The Reason-Phrase should identify the syntax problem in more detail, for example, “Missing Call-ID header field”.

IPEC_SIPReasonStatus401Unauthorized (0x1519, 5401 decimal)
SIP Request Failure Response 401 - Unauthorized - The request requires user authentication. This response is issued by User Agent Servers (UASs) and registrars, while 407 (Proxy Authentication Required) is used by proxy servers.

IPEC_SIPReasonStatus402PaymentRequired (0x151a, 5402 decimal)
SIP Request Failure Response 402 - Payment Required - Reserved for future use.

IPEC_SIPReasonStatus403Forbidden (0x151b, 5403 decimal)
SIP Request Failure Response 403 - Forbidden - The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request should not be repeated.

IPEC_SIPReasonStatus404NotFound (0x151c, 5404 decimal)
SIP Request Failure Response 404 - Not Found - The server has definitive information that the user does not exist at the domain specified in the Request-URI. This status is also returned if the domain in the Request-URI does not match any of the domains handled by the recipient of the request.

IPEC_SIPReasonStatus405MethodNotAllowed (0x151d, 5405 decimal)
SIP Request Failure Response 405 - Method Not Allowed - The method specified in the Request-Line is understood, but not allowed for the address identified by the Request-URI. The response must include an Allow header field containing a list of valid methods for the indicated address.

IPEC_SIPReasonStatus406NotAcceptable (0x151e, 5406 decimal)
SIP Request Failure Response 406 - Not Acceptable - The resource identified by the request is only capable of generating response entities that have content characteristics not acceptable according to the Accept header field sent in the request.

IPEC_SIPReasonStatus407ProxyAuthenticationRequired (0x151f, 5407 decimal)
SIP Request Failure Response 407 - Proxy Authentication Required - This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. This status code can be used for applications where access to the communication channel (for example, a telephony gateway) rather than the callee, requires authentication.

IPEC_SIPReasonStatus408RequestTimeout (0x1520, 5408 decimal)
SIP Request Failure Response 408 - Request Timeout - The server could not produce a response within a suitable amount of time, for example, if it could not determine the location of the user in time. The client may repeat the request without modifications at any later time.

IPEC_SIPReasonStatus410Gone (0x1522, 5410 decimal)
SIP Request Failure Response 410 - Gone - The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) should be used instead.

IPEC_SIPReasonStatus413RequestEntityTooLarge (0x1525, 5413 decimal)
SIP Request Failure Response 413 - Request Entity Too Large - The server is refusing to process a request because the request entity-body is larger than the server is willing or able to process. The server may close the connection to prevent the client from continuing the request. If the condition is temporary, the server should include a Retry-After header field to indicate that it is temporary and after what time the client may try again.

IPEC_SIPReasonStatus414RequestUriTooLong (0x1526, 5414 decimal)
SIP Request Failure Response 414 - Request-URI Too Long - The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

IPEC_SIPReasonStatus415UnsupportedMediaType (0x1527, 5415 decimal)
SIP Request Failure Response 415 - Unsupported Media Type - The server is refusing to service the request because the message body of the request is in a format not supported by the server for the requested method. The server must return a list of acceptable formats using the Accept, Accept-Encoding, or Accept-Language header field, depending on the specific problem with the content.

IPEC_SIPReasonStatus416UnsupportedURIScheme (0x1528, 5416 decimal)
SIP Request Failure Response 416 - Unsupported URI Scheme - The server cannot process the request because the scheme of the URI in the Request-URI is unknown to the server.

IPEC_SIPReasonStatus420BadExtension (0x153c, 5420 decimal)
SIP Request Failure Response 420 - Bad Extension - The server did not understand the protocol extension specified in a Proxy-Require or Require header field. The server must include a list of the unsupported extensions in an Unsupported header field in the response.

IPEC_SIPReasonStatus421ExtensionRequired (0x153d, 5421 decimal)
SIP Request Failure Response 421 - Extension Required - The User Agent Server (UAS) needs a particular extension to process the request, but this extension is not listed in a Supported header field in the request. Responses with this status code must contain a Require header field listing the required extensions. A UAS should not use this response unless it truly cannot provide any useful service to the client. Instead, if a desirable extension is not listed in the Supported header field, servers should process the request using baseline SIP capabilities and any extensions supported by the client.

IPEC_SIPReasonStatus423IntervalTooBrief (0x153f, 5423 decimal)
SIP Request Failure Response 423 - Interval Too Brief - The server is rejecting the request because the expiration time of the resource refreshed by the request is too short. This response can be used by a registrar to reject a registration whose Contact header field expiration time was too small.

IPEC_SIPReasonStatus480TemporarilyUnavailable (0x1568, 5480 decimal)
SIP Request Failure Response 480 - Temporarily Unavailable - The callee's end system was contacted successfully but the callee is currently unavailable (for example, is not logged in, logged in but in a state that precludes communication with the callee, or has activated the “do not disturb” feature). The response may indicate a better time to call in the Retry-After header field. The user could also be available elsewhere (unbeknownst to this server). The reason

phrase should indicate a more precise cause as to why the callee is unavailable. This value should be settable by the User Agent (UA). Status 486 (Busy Here) may be used to more precisely indicate a particular reason for the call failure. This status is also returned by a redirect or proxy server that recognizes the user identified by the Request-URI, but does not currently have a valid forwarding location for that user.

IPEC_SIPReasonStatus481CallTransactionDoesNotExist (0x1569, 5481 decimal)
SIP Request Failure Response 481 - Call/Transaction Does Not Exist - This status indicates that the User Agent Server (UAS) received a request that does not match any existing dialog or transaction.

IPEC_SIPReasonStatus482LoopDetected (0x156a, 5482 decimal)
SIP Request Failure Response 482 - Loop Detected - The server has detected a loop.

IPEC_SIPReasonStatus483TooManyHops (0x156b, 5483 decimal)
SIP Request Failure Response 483 - Too Many Hops - The server received a request that contains a Max-Forwards header field with the value zero.

IPEC_SIPReasonStatus484AddressIncomplete (0x156c, 5484 decimal)
SIP Request Failure Response 484 - Address Incomplete - The server received a request with a Request-URI that was incomplete. Additional information should be provided in the reason phrase. This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a 484 (Address Incomplete) status response.

IPEC_SIPReasonStatus485Ambiguous (0x156d, 5485 decimal)
SIP Request Failure Response 485 - The Request-URI was ambiguous. The response may contain a listing of possible unambiguous addresses in Contact header fields. Revealing alternatives can infringe on privacy of the user or the organization. It must be possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of possible choices for ambiguous Request-URIs.

IPEC_SIPReasonStatus486BusyHere (0x156e, 5486 decimal)
SIP Request Failure Response 486 - Busy Here - The callee's end system was contacted successfully, but the callee is currently not willing or able to take additional calls at this end system. The response may indicate a better time to call in the Retry-After header field. The user could also be available elsewhere, such as through a voice mail service. Status 600 (Busy Everywhere) should be used if the client knows that no other end system will be able to accept this call.

IPEC_SIPReasonStatus487RequestTerminated (0x156f, 5487 decimal)
SIP Request Failure Response 487 - Request Terminated - The request was terminated by a BYE or CANCEL request. This response is never returned for a CANCEL request itself.

IPEC_SIPReasonStatus488NotAcceptableHere (0x1570, 5488 decimal)
SIP Request Failure Response 488 - Not Acceptable Here - The response has the same meaning as 606 (Not Acceptable), but only applies to the specific resource addressed by the Request-URI and the request may succeed elsewhere. A message body containing a description of media capabilities may be present in the response, which is formatted according to the Accept header field in the INVITE (or application/SDP if not present), the same as a message body in a 200 (OK) response to an OPTIONS request.

IPEC_SIPReasonStatus491RequestPending (0x1573, 5491 decimal)
SIP Request Failure Response 491 - Request Pending - The request was received by a User Agent Server (UAS) that had a pending request within the same dialog.

IPEC_SIPReasonStatus493Undecipherable (0x1575, 5493 decimal)

SIP Request Failure Response 493 - Undecipherable - The request was received by a User Agent Server (UAS) that contained an encrypted MIME body for which the recipient does not possess or will not provide an appropriate decryption key. This response may have a single body containing an appropriate public key that should be used to encrypt MIME bodies sent to this User Agent (UA).

Server Failure Response Codes (5xx)

IPEC_SIPReasonStatus500ServerInternalError (0x157c, 5500 decimal)

Server Failure Response 500 - Server Internal Error - The server encountered an unexpected condition that prevented it from fulfilling the request. The client may display the specific error condition and may retry the request after several seconds. If the condition is temporary, the server may indicate when the client may retry the request using the Retry-After header field.

IPEC_SIPReasonStatus501NotImplemented (0x157d, 5501 decimal)

Server Failure Response 501 - Not Implemented - The server does not support the functionality required to fulfill the request. This is the appropriate response when a User Agent Server (UAS) does not recognize the request method and is not capable of supporting it for any user. Proxies forward all requests regardless of method. Note that a 405 (Method Not Allowed) is sent when the server recognizes the request method, but that method is not allowed or supported.

IPEC_SIPReasonStatus502BadGateway (0x157e, 5502 decimal)

Server Failure Response 502 - Bad Gateway - The server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfill the request.

IPEC_SIPReasonStatus503ServiceUnavailable (0x157f, 5503 decimal)

Server Failure Response 503 - Service Unavailable - The server is temporarily unable to process the request due to a temporary overloading or maintenance of the server. The server may indicate when the client should retry the request in a Retry-After header field. If no Retry-After is given, the client must act as if it had received a 500 (Server Internal Error) response. A client (proxy or User Agent Client) receiving a 503 (Service Unavailable) should attempt to forward the request to an alternate server. It should not forward any other requests to that server for the duration specified in the Retry-After header field, if present. Servers may refuse the connection or drop the request instead of responding with 503 (Service Unavailable).

IPEC_SIPReasonStatus504ServerTimeout (0x1580, 5504 decimal)

Server Failure Response 504 - Server Time-out - The server did not receive a timely response from an external server it accessed in attempting to process the request. 408 (Request Timeout) should be used instead if there was no response within the period specified in the Expires header field from the upstream server.

IPEC_SIPReasonStatus505VersionNotSupported (0x1581, 5505 decimal)

Server Failure Response 505 - Version Not Supported - The server does not support, or refuses to support, the SIP protocol version that was used in the request. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message.

IPEC_SIPReasonStatus513MessageTooLarge (0x1589, 5513 decimal)

Server Failure Response 513 - Message Too Large - The server was unable to process the request since the message length exceeded its capabilities.

Global Failure Response Codes (6xx)

IPEC_SIPReasonStatus600BusyEverywhere (0x15e0, 5600 decimal)

SIP Global Failure Response 600 - Busy Everywhere - The callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time. The response may indicate a better time to call in the Retry-After header field. If the callee does not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead. This status response is returned only if the client knows that no other end point (such as a voice mail system) will answer the request. Otherwise, 486 (Busy Here) should be returned.

IPEC_SIPReasonStatus603Decline (0x15e3, 5603 decimal)

SIP Global Failure Response 603 - 603 Decline - The callee's machine was successfully contacted but the user explicitly does not wish to or cannot participate. The response may indicate a better time to call in the Retry-After header field. This status response is returned only if the client knows that no other end point will answer the request.

IPEC_SIPReasonStatus604DoesNotExistAnywhere (0x15e4, 5604 decimal)

SIP Global Failure Response 604 - Does Not Exist Anywhere - The server has authoritative information that the user indicated in the Request-URI does not exist anywhere.

IPEC_SIPReasonStatus606NotAcceptable (0x15e6, 5606 decimal)

SIP Global Failure Response 606 - Not Acceptable - The user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable. A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately support the session described.

The 606 (Not Acceptable) response may contain a list of reasons in a Warning header field describing why the session described cannot be supported.

A message body containing a description of media capabilities may be present in the response, which is formatted according to the Accept header field in the INVITE (or application/SDP if not present), the same as a message body in a 200 (OK) response to an OPTIONS request.

It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide whether or not to act on a 606 (Not Acceptable) response.

This status response is returned only if the client knows that no other end point will answer the request.

Other SIP Codes (8xx)

IPEC_SIPReasonStatusBYE (0x16a8, 5800 decimal)

SIP reason status 800. BYE code.

IPEC_SIPReasonStatusCANCEL (0x16a9, 5801 decimal)

SIP reason status 801. CANCEL code.

Supplementary Reference Information 11

This chapter lists related publications and includes other reference information as follows:

11.1 References to More Information

The following publications provide related information:

- ITU-T Recommendation H.323 (11/00) - Packet-based multimedia communications systems
- ITU-T Recommendation H.245 (07/01) - Control protocol for multimedia communication
- ITU-T Recommendation H.225.0 (09/99) - Call signaling protocols and media stream packetization for packet-based multimedia communications systems
- ITU-T Recommendation T.38 (06/98) - Procedures for real-time Group 3 facsimile communication over IP networks
- ITU-T Recommendation T.30 (07/96) - Procedures for document facsimile transmission in the general switched telephone network
- RFC 1889, *RTP: A Transport Protocol for Real-Time Applications*, IETF Publication, *http://www.ietf.org/rfc/rfc1889.txt*
- RFC 3261, *Session Initiation Protocol (SIP)*, IETF Publication, draft reference *http://www.ietf.org/rfc/rfc3261.txt?number=3261*
- Cisco Systems, *Signaled Digits in SIP*, draft reference *http://www.ietf.org/internet-drafts/draft-mahy-sipping-signaled-digits-00.txt*
- Black, Uyless, *Voice over IP*, Prentice Hall PTR, Prentice-Hall, Inc. (Copyright 2000)
- Douskalis, Bill, *IP Telephony; The Integration of Robust VoIP Services*, Prentice Hall PTR, Prentice-Hall, Inc., ISBN 0-13-014118-6
- Galtieri, Paolo, *Introduction to Voice Over the Internet Protocol*, Applied Computing Technologies, Winter 2000

11.2 Called and Calling Party Address List Format When Using H.323

This section provides reference information about called and calling party address list format:

- Called Party Address List
- Calling Party Address List
- Examples of Called and Calling Party Addresses

Called Party Address List

Called party address lists are formatted 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):
  - **0.PUU** - The numbering plan follows the E.163 and E.164Recommendations.
  - **PUI** - The number digits carry a prefix indicating type of number according to national recommendations.
  - **PUN** - The number digits carry a prefix indicating the type of number according to national recommendations.
  - **PUNS** - The number digits carry a prefix indicating the type of number according to network specifications.
  - **PUA** - Valid only for the called party number at the outgoing access; the network substitutes appropriate number.
  - **D** - Valid only for the called party number at the outgoing access; the network substitutes appropriate number.
  - **PRL2** - Level 2 regional subtype of private number.
  - **PRL1** - Level 1 regional subtype of private number.
  - **PRP** - PISN subtype of private number.
  - **PRL** - Local subtype of private number.
  - **PRA** - Abbreviated subtype of private number.
  - **N** - 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 by setting the value of the delimiter field in the IPCCLIB_START_DATA used by the **gc_Start( )** function. See Section 7.2.26, "gc_Start( ) Variances for IP", on page 201 for more information.

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

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 Called Party Address List section.

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.

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.

blind transfer: See *unsupervised transfer.*

call transfer: See *supervised transfer* and *unsupervised transfer.*

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.255.0: The H.255.0 standard defines a layer that formats the transmitted audio, video, data, and control streams for output to the network, and retrieves the corresponding streams from the network.

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: Intel API 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).

supervised transfer: A call transfer in which the person transferring the call stays on the line, announces the call, and consults with the party to whom the call is being transferred before the transfer is completed.

UA: In a SIP context, user agents (UAs) are appliances or applications, such as, SIP phones, residential gateways and software that initiate and receive calls over a SIP network.

SIP: Session Initiated Protocol. An ASCII-based, peer-to-peer protocol designed to provide telephony services over the Internet.

split call control: An IP telephony software architecture in which call control is done separately from IP Media stream control, for example, call control is done on the host and IP Media stream control is done on the board.

tunneling: The encapsulation of H.245 messages within Q.931/H.225 messages so that H.245 media control messages can be transmitted over the same TCP port as the Q.931/H.225 signaling messages.

unsupervised transfer: A transfer in which the call is transferred without any consultation or announcement by the person transferring the call.

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

I

L

M

N

O

P

V