Dialogic

Dialogic® Host Media Processing Software Release 3.1LIN

Release Update

February 25, 2011

05-2452-027

Publication Date: February 25, 2011

Document Number: 05-2452-027

About This Publication

This section contains information about the following topics:

Purpose

This Release Update addresses issues associated with the Dialogic® Host Media Processing Software Release 3.1LIN (also referred to as Dialogic® HMP Software 3.1LIN). In addition to summarizing issues that were known as of this release, it is intended that the Release Update will continue to be updated to serve as the primary mechanism for communicating any new issues that may arise after the release date.

Intended Audience

This Release Update is intended for all users of Dialogic® Host Media Processing Software Release 3.1LIN.

How to Use This Publication

This Release Update is organized into the following sections (click the section name to jump to the corresponding section):

- Document Revision History: This section summarizes the ongoing changes and additions that are made to this Release Update after its original release. This section is organized by document revision and document section.
- Post-Release Developments: This section describes significant changes to the release subsequent to the general availability release date. For example, new features provided in the Service Update are described in this section.
- Release Issues: This section lists issues that may affect the system release hardware and software.
- Documentation Updates: This section contains corrections and other changes that apply to the documentation not made prior to the release. These updates are organized by documentation category and by individual document.

Related Information

See the following for additional information:

- For information about the products and features supported in this release, see the *Dialogic® Host Media Processing Software Release 3.1LIN Release Guide*, which is included as part of the documentation bookshelf for the release.
- For further information on issues that have an associated defect number, you may use the Defect Tracking tool at *http://membersresource.dialogic.com/defects/*. When you select this link, you will be asked to either LOGIN or JOIN.
- *http://www.dialogic.com/manuals/* (for Dialogic® product documentation)
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

Document Revision History

This Revision History summarizes the changes made in this and each previously published version of the Release Update for Dialogic® Host Media Processing Software Release 3.1LIN, which is a document that is planned to be periodically updated throughout the lifetime of the release.

Document Rev 27 - February 25, 2011

Updated for Service Update 246.

In the Post-Release Developments chapter, added:

- GCC 4.1.1 Compiler Support.
- dx_setchxfercnt( ) API Function Support.

In the Documentation Updates chapter:

- Update to the “Supported Compilers” subsection in the Dialogic® Host Media Processing Software Release 3.1LIN Release Guide.
- Updated text in the “Cleaning the RTF Log File” section of the Dialogic® Host Media Processing Diagnostics Guide.
- Added multiprocessing information to the “Performance Considerations” section of the Dialogic® Standard Runtime Library API Programming Guide.

Document Rev 26 - December 17, 2010

Updated for Service Update 244.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00047881, IPY00054971, IPY00055853, IPY00055887, IPY00055976, IPY00056010, IPY00056033, IPY00056036, IPY00081118, IPY00081808, IPY00091103, IPY00091419, IPY00091502, IPY00091783, IPY00091867, IPY00091910, IPY00091918, IPY00091959, IPY00091982, IPY00092084, IPY00092200, IPY00092209, IPY00092243, IPY00092278, IPY00092280, IPY00092285, IPY00092357, IPY00092395.

In the Documentation Updates chapter:

- Removed documentation updates to the Dialogic® Host Media Processing Software for Linux Configuration Guide because a new version is available on the bookshelf. This guide was previously named the Dialogic® System Configuration Guide.
- Removed documentation updates to the MSML Media Server Software User’s Guide because a new version is available on the bookshelf.

Document Rev 25 - August 27, 2010

Updated for Service Update 240.

In the Post-Release Developments chapter, added:

- Unspecified G.723.1 Bit Rate in Outgoing SIP Requests with SDP.
- Overlap-Receive Support for Limited SIP-I Interworking Scenarios.
- Processing Multiple 18x Provisional Responses.
- TLS and SRTP Channel Support Increase.
- 33 Frames Per Packet Support (AMR).
- Registering Authentication Data without Realm String.
- Handling non-2xx Responses to T.38 Switch.
- MIME Insertion in Outgoing ACK.
- Increase in Channel Density for G.711 Codec.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00055439, IPY00055506, IPY00055507, IPY00055625, IPY00079689, IPY00080124, IPY00080125, IPY00081685, IPY00081732, IPY00082125, IPY00082294, IPY00082301, IPY00082344, IPY00090645, IPY00090646, IPY00090750, IPY00090816, IPY00090932, IPY00090994, IPY00091023, IPY00091091, IPY00091093, IPY00091132, IPY00091138, IPY00091139, IPY00091140, IPY00091162, IPY00091292, IPY00091348, IPY00091361, IPY00091383, IPY00091529.

In the Documentation Updates chapter:

- Added updates to the Dialogic® Host Media Processing Software for Linux Configuration Guide due to an increase in frames per packet for AMR, and to resolve IPY00091436.

Document Rev 24 - May 7, 2010

Updated for Service Update 237.

In the Post-Release Developments chapter, added:

- Monitor Mode Support for HMP Conferencing.
- 3PCC Support for Dynamic Selection of Outbound SIP Proxy.
- An update to content as a result of adding 3PCC support in Support for Dynamic Selection of Outbound SIP Proxy.
- Updates to the diagrams in Defer the Sending of SIP Messages.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00055290, PY00080472, IPY00081235, IPY00081664, IPY00081676, IPY00082088, IPY00082163, IPY00082165.

In the Documentation Updates chapter:

- Update the range for QOSTYPE_RTPTIMEOUT in the Dialogic® IP Media Library API Programming Guide and Library Reference. (IPY00090740)

Document Rev 23 - January 15, 2010

Updated for Service Update 233.

In the Post-Release Developments chapter:

- Added Support for Dynamic Selection of Outbound SIP Proxy.
- Added Defer the Sending of SIP Messages.
- Added Support for WaitCall Cancellation.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00054969, IPY00054970, IPY00055020, IPY00080727, IPY00081132, IPY00081264, IPY00081284, IPY00081399, IPY00081410, IPY00081489, IPY00081571, IPY00081661, IPY00081665, IPY00081756.

In the Documentation Updates chapter:

- Update to Quality of Service (QoS) parameter defaults in the Dialogic® IP Media Library API Programming Guide and Library Reference. (IPY00080727)
- Update to Section 2.2, “mediaserver Configuration Script” in the MSML Media Server Software User’s Guide. (IPY00080612)

Document Rev 22 - November 11, 2009

Updated for Service Update 229.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00081107, IPY00081126, IPY00081211 IPY00081254, IPY00081277.
- Added the following Known Issue related to SU 227 only: IPY00081277.

In the Documentation Updates chapter:

- A new version of the Dialogic® Host Media Processing Software for Linux Configuration Guide is available on the documentation bookshelf.
- Deleted the updates for the Dialogic® Host Media Processing Diagnostics Guide, because these updates have been incorporated into the revised document that is now on the online documentation bookshelf.
- A new version of the Dialogic® Device Management API Library Reference is available on the documentation bookshelf.
- A new version of the Dialogic® IP Media Library API Programming Guide and Library Reference is available on the documentation bookshelf.

- A new version of the Dialogic® Multimedia API Programming Guide and Library Reference is available on the documentation bookshelf.

Document Rev 21 - September 16, 2009

Updated for Service Update 227.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00080658, IPY00080790.

In the Documentation Updates chapter:

- Updated a software download path in the Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide.

Document Rev 20 - published July 2, 2009

Updated for Service Update 226.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00079344, IPY00079346, IPY00079375, IPY00079435, IPY00079734, IPY00079742, IPY00079768, IPY00079787, IPY00079825, IPY00079866, IPY00079897, IPY00079902, IPY00079929, IPY00079931, IPY00079981, IPY00080129, IPY00080130, IPY00080297, IPY00080324, IPY00080336, IPY00080364, IPY00080471, IPY00080556, IPY00080564, IPY00080570, IPY00080622.

Document Rev 19 - published April 24, 2009

Updated for Service Update 223.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00045089, IPY00045524, IPY00046378, IPY00078309, IPY00078344, IPY00078369, IPY00078440, IPY00079108, IPY00079137, IPY00079186, IPY00079201, IPY00079251, IPY00079374, IPY00079393, IPY00079466, IPY00079483, IPY00079502, IPY00079509, IPY00079518, IPY00079530, IPY00079551, IPY00079639.

In the Documentation Updates chapter:

- Deleted the updates for the Dialogic® Voice API Library Reference, because these updates have been incorporated into the revised document that is now on the online documentation bookshelf.
- Deleted the updates for the Dialogic® Voice API Programming Guide, because these updates have been incorporated into the revised document that is now on the online documentation bookshelf.

Document Rev 18 - published February 27, 2009

Updated for Service Update 221.

In the Post-Release Developments chapter:

- Added Support for Enhanced RTCP Reports.
- Added Support for Streaming H.263 Baseline Profile Using RFC 2429 Packetization.
- Added AMR-NB and G.711 Audio Over Nb UP.
- Added Support for Red Hat Enterprise Linux Release 5.0 Update 2.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00042207, IPY00043498, IPY00044586, IPY00044901, IPY00044997, IPY00078397, IPY00078438, IPY00078440, IPY00078474, IPY00078791, IPY00078792, IPY00078793, IPY00078800, IPY00079023, IPY00079132, IPY00079506.

In the Documentation Updates chapter:

- Update to the Dialogic® Host Media Processing Software Release 3.1LIN Release Guide to show support for Red Hat Enterprise Linux Release 5.0 Update 2.
- Added that a new version of Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide is now available on the documentation bookshelf.
- Added that a new version of Dialogic® 3G-324M API Programming Guide and Library Reference is now available on the documentation bookshelf.
- Added that a new version of Dialogic® Device Management API Library Reference is now available on the documentation bookshelf.
- Added that a new version of Dialogic® Global Call E1/T1 CAS/R2 Technology Guide is now available on the documentation bookshelf.
- Added that a new version of Dialogic® Global Call ISDN Technology Guide is now available on the documentation bookshelf.
- Added that a new version of Dialogic® Global Call SS7 Technology Guide is now available on the documentation bookshelf.
- Added that a new version of Dialogic® IP Media Library API Programming Guide and Library Reference is now available on the documentation bookshelf.
- Added that a new version of Dialogic® Multimedia API Programming Guide and Library Reference is now available on the documentation bookshelf.
- Added that a new version of MSML Media Server Software User's Guide is now available on the documentation bookshelf.
- Added that a new version of Dialogic® 3G-324M Multimedia Gateway Demo Guide is now available on the documentation bookshelf.

Document Rev 17 - published October 31, 2008

Updated for Service Update 210.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00045033, IPY00045077, IPY00045346, IPY00045407, IPY00078369, IPY00078392.

Document Rev 16 - published October 3, 2008

Updated for Service Update 204.

In the Post-Release Developments chapter:

- Added Support for Red Hat Enterprise Linux Release 5.0 Advanced Server, Enterprise Server, or Workstation (AS, ES, or WS).

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00044224, IPY00044285, IPY00044750, IPY00044834, IPY00044978, IPY00045006, IPY00045156.
- Added the following Known Issues: IPY00044980.

In the Documentation Updates chapter:

- Added update to Dialogic® Standard Runtime Library API Library Reference for sr_getfdcnt( ) and sr_getfdinfo( ) functions (IPY00045054).

Document Rev 15 - published September 22, 2008

Updated for Service Update 203.

In the Post-Release Developments chapter:

- Added Enabling Runtime Tracing in 3G-324M API.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00042913, IPY00043399, IPY00043836, IPY00043860, IPY00043940, IPY00044125, IPY00044219, IPY00044235, IPY00044273, IPY00044447, IPY00044488, IPY00044509, IPY00044544, IPY00044578, IPY00044633, IPY00044700, IPY00044926.

In the Documentation Updates chapter:

- Added that new version of Dialogic® Host Media Processing Software for Linux Configuration Guide is now available on the documentation bookshelf.
- Added that new version of Dialogic® IP Media Library API Programming Guide and Library Reference is now available on the documentation bookshelf.

Document Rev 14 - published August 25, 2008

Updated for Service Update 201.

In the Post-Release Developments chapter:

- Added Multiple Frames Per Packet Support for AMR-NB.

In the Release Issues chapter:

- Added the following Known Permanent Issue: IPY00044550.

In the Documentation Updates chapter:

- Added that new version of Dialogic® Host Media Processing Software Release 3.1LIN Release Guide is now available on the documentation bookshelf.
- Added update to Dialogic® IP Media Library API Programming Guide and Library Reference for IPM_AUDIO_CODER_INFO data structure.
- Added update to Dialogic® Voice API Library Reference for dx_OpenStreamBuffer( ) function.

Document Rev 13 - published June 30, 2008

Updated for Service Update 197.

In the Post-Release Developments chapter:

- Added Support for ATDX_BUFDIGS( ) Function.

In the Release Issues chapter:

- Added the following Resolved Issue: IPY00044118.

In the Documentation Updates chapter:

- Added that new version of Dialogic® Host Media Processing Software Release 3.1LIN Release Guide is now available on the documentation bookshelf.
- Added update to Dialogic® Global Call API Library Reference for gc_util_insert_parm_val( ) function (IPY00043078).
- Added updates to Dialogic® Global Call SS7 Technology Guide.
- Added update to Dialogic® IP Media Library API Programming Guide and Library Reference for unCoderPayloadType field of IPM_AUDIO_CODER_INFO data structure (IPY00044398).
- Added update to Dialogic® Voice API Library Reference for ATDX_BUFDIGS( ) function.

Document Rev 12 - published May 30, 2008

Updated for Service Update 192.

In the Post-Release Developments chapter:

- Added Retrieving FlexLM-based Licensed Feature Data.
- Added AMD Machine Support for IP-only Configurations.

In the Release Issues chapter:

- Added the following Resolved Issue: IPY00043038. Also added Resolved Issue (no defect number) regarding potential application crash when running conferencing at max density.

Document Rev 11 - published May 9, 2008

Updated for Service Update 190.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00042917, IPY00042933, IPY00043023, IPY00043133, IPY00043214, IPY00043216.

In the Documentation Updates chapter:

- Added updates to Dialogic® IP Media Library API Programming Guide and Library Reference for IPM_MEDIA and IPM_VIDEO_CODER_INFO data structures.

Document Rev 10 - published April 22, 2008

Additional Update for Service Update 187.

In the Post-Release Developments chapter:

- Added Support for Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards.
- Added the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards to the list of supported boards in Enabling Echo Cancellation with Non-Linear Processing.

In the Documentation Updates chapter:

- Provided details of the changes made to the M3G_PARM_INFO structure reference page in Dialogic® 3G-324M API Programming Guide and Library Reference.

Document Rev 09 - published April 16, 2008

Updated for Service Update 187.

In the Post-Release Developments chapter:

- Added Revised Multimedia User I/O Implementation.
- Added Additional Operating System Support.
- Added Receive-only RFC2833 Mode Available.
- Removed details from Support for SS7 Functionality as that information is now available in the new version of Dialogic® Global Call SS7 Technology Guide.
- Added Service Update numbers to each feature description to identify the release where the feature was introduced.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00042216, IPY00042519, IPY42716.

In the Documentation Updates chapter:

- Removed documentation updates as new version of Dialogic® Host Media Processing Software Release 3.1LIN Release Guide is now available on the documentation bookshelf.
- Removed documentation updates as new version of Dialogic® Global Call SS7 Technology Guide is now available on the documentation bookshelf.
- Updates to the Data Structure chapter and the M3G_PARM_INFO structure reference page in Dialogic® 3G-324M API Programming Guide and Library Reference.
- Removed documentation updates as new version of Dialogic® Multimedia API Programming Guide and Library Reference is now available on the documentation bookshelf.

Document Rev 08 - published February 7, 2008

Updated for Service Update 174.

In the Post-Release Developments chapter:

- Added Support for GSM Codec.
- Renamed the section from Support for MSML Video to Support for MSML Video, PSTN, and MSML Audit Package since PSTN and Audit Package are also supported.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00041064, IPY00041301, IPY00041508, IPY00041542.
- Added the following Known Issue: IPY00041750.

In the Documentation Updates chapter:

- Added updates to Dialogic® Host Media Processing Software Release 3.1LIN Release Guide, Dialogic® Voice API Library Reference, and Dialogic® Voice API Programming Guide for GSM Codec support.
- Added update to Dialogic® Host Media Processing Software Release 3.1LIN Release Guide for increased channel density support on 3G-324M.

Document Rev 07 - published January 11, 2008

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00041320, IPY00041454, IPY00041590.

In the Documentation Updates chapter:

- Added corrections to the parameter and event descriptions in Dialogic® Multimedia API Programming Guide and Library Reference.

Document Rev 06 - published November 16, 2007

Updated for Service Update 157.

In the Post-Release Developments chapter:

- Added Support for 3G-324M Release 4 (Nb UP).
- Added Support for SUSE Linux Enterprise Server 9 Service Pack 3.
- Added Support for Dialogic® SS7 Boards.
- Added Support for MSML Video, PSTN, and MSML Audit Package.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00039666, IPY00039853, IPY00040018, IPY00041041.
- Added the following Known Issue: IPY00041133.

In the Documentation Updates chapter:

- Added updates to Dialogic® Host Media Processing Software Release 3.1LIN Release Guide for 3G-324M Release 4 (Nb UP) support, SUSE Linux Enterprise Server 9 Service Pack 3 support, Dialogic® SS7 Boards support, and MSML Video support.

Document Rev 05 - published November 1, 2007

Updated for Service Update 154.

In the Post-Release Developments chapter:

- Added Support for Dialogic® DNI2410TEPEHMP Digital Network Interface Board.
- Added Enabling Echo Cancellation with Non-Linear Processing.

In the Release Issues chapter:

- Added the following Known Issues: IPY00039666, IPY00039853, IPY00039990, IPY00040471, IPY00040483, IPY00040928, IPY00041064.

In the Documentation Updates chapter:

- Added a sentence about compiling applications to the following programming guides: Dialogic® Audio Conferencing API Programming Guide, Dialogic® Conferencing API Programming Guide, Dialogic® Continuous Speech Processing API Programming Guide, Dialogic® Global Call API Programming Guide, Dialogic® IP Media Library API Programming Guide and Library Reference, Dialogic® Voice API Programming Guide.

Document Rev 04 - published October 17, 2007

In the Documentation Updates chapter:

- Added table in Channel Density Support section for Dialogic® Host Media Processing Software Release 3.1LIN Release Guide that lists higher channel density ranges and supported resources.
- Added that new version of document is available in Dialogic® Host Media Processing Software Release 3.1LIN Release Guide.
- Added update to Section 3.1, Obtaining a License for Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide.

Document Rev 03 - published September 17, 2007

In the Documentation Updates chapter:

- Added update to Dialogic® Host Media Processing Software for Linux Configuration Guide for details on increasing the number of message queues.

Document Rev 02 - published September 12, 2007

Updated for Service Update 139.

In the Post-Release Developments chapter:

- Added Service Update for Dialogic® HMP Software 3.1LIN.
- Added Support for SS7 Functionality.

In the Release Issues chapter:

- Added the following Resolved Issues: IPY00039715, IPY00039825, IPY00039858.

In the Documentation Updates chapter:

- Added a new document named Dialogic® Global Call SS7 Technology Guide to support SS7 functionality.

Document Rev 01 - published August 27, 2007

Initial version of document.

Post-Release Developments 1

This section describes significant changes to Dialogic® Host Media Processing Software Release 3.1LIN subsequent to the general availability release.

1.1 Service Update for Dialogic® HMP Software 3.1LIN

A Service Update for Dialogic® HMP Software 3.1LIN is now available. Service Updates provide fixes to known problems, and may also introduce new functionality. New versions of the Service Update will be released periodically. It is intended that this Release Update will document the features in the Service Updates.

1.2 GCC 4.1.1 Compiler Support

Service Update 246 fully supports GNU Compiler Collection (GCC) 4.1.1. Now, all Dialogic libraries are fully linked to the GCC 4.1.1 libraries.

1.3 dx_setchxfercnt( ) API Function Support

With this Service Update 246, the Voice API function, **dx_setchxfercnt( )**, is supported with the Dialogic® HMP 3.1LIN Software Release.

1.3.1 Function Information

The **dx_setchxfercnt( )** function sets the bulk queue buffer size for a channel. It is typically used in conjunction with the user I/O feature or the streaming to board feature.

For details about this function, refer to the *Dialogic® Voice API Library Reference* available on the documentation bookshelf.

Note: When using UIO with bulk queue buffer sizes of 4 Kbytes or more (bufsize_identifier is set to 0 to 3), the number of initial consecutive callbacks invoked during a playback depends on the buffer size used. To avoid underrun conditions, it is recommended that you have a minimum of 16 Kbytes of data required to successfully start playback or streaming to board.

When using UIO with bulk queue buffer sizes of 2 Kbytes or less (bufsize_identifier is set to 4, 5, or 6), the application UIO routine will get invoked six to seven consecutive times (with no delay) initially for a given buffer size during a playback. This means that a sufficient amount of data must be available for the entire sequence of consecutive invocations for successful playback of that data. When using streaming to board, the application should prime the circular buffer with at least six times the data required to play the bulk queue buffer size to avoid an underrun condition during the playback.

The minimum transfer buffer size supported on a **recording** operation is 4 Kbytes (bufsize_identifier=0); setting the value to anything lower than that will yield 4 Kbytes.

1.4 Unspecified G.723.1 Bit Rate in Outgoing SIP Requests with SDP

With Service Update 240, a Global Call application in 1PCC mode can choose not to specify the G.723.1 codec bit rate, namely 5.3 kbps or 6.3 kbps, in an outgoing SIP message with SDP body. Instead, the application can let the far end UA request the bit rate. Feature enablement and disablement can be controlled either at the IPT board-level device or the IPT network device (channel).

1.4.1 Feature Overview

G.723.1 is a dual-rate codec supporting 5.3 kbps and 6.3 kbps bit rates. With SIP, the choice of one or the other rate for RTP streaming is specified in an SDP body in outgoing SIP messages using an optional bit rate parameter in the format transport (a=fmtp) attribute of the media announcement. Currently, when in 1PCC mode, Global Call always includes this optional bit rate parameter in SDP bodies irrespective of whether the application's IP_CAPABILITY data structure setting specifies a single G.723.1 bit rate (5.3 or 6.3 kbps), both G.723.1 bit rates (5.3 and 6.3 kbps), or "don't care" for local transmit rates.

When both G.723.1 rates are specified, only the one in the first IP_CAPABILITY structure is included. The following example shows an SDP message extract from an HMP SIP request where the application set both rates in the IP_CAPABILITY data structure. The outgoing SDP contains only the bit rate that was specified first. In this particular case, the capability field in the first IP_CAPABILITY element with IP_CAP_DIR_LCLTRANSMIT direction was set to GCCAP_AUDIO_g7231_5_3k for the IPT network device:

```
v=0
o=Intel_IPCCLib 50083120 50083121 IN IP4 192.168.185.64
s=Dialogic_SIP_CCLLIB
i=session information
c=IN IP4 192.168.185.64
t=0 0
m=audio 49152 RTP/AVP 0 8 4 101
a=rtpmap:4 G723/8000
a=fmtp:4 bitrate=5.3; annexa=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

With this feature, when the Global Call 1PCC application specifies both G.723.1 bit rates as transmit codec selection, the HMP software no longer sends the optional bit rate parameter in the SDP body of a SIP request or response. Instead, the remote UA has the responsibility to select one of the G.723.1 bit rates for bit rate negotiation.

Note: If only one of the G.723.1 bit rates is specified by the application transmit codec selection, the HMP software maintains its existing behavior and the specified selection applies when negotiating the bit rate to use irrespective of this feature enablement. Likewise, there is no change in behavior if the application selects “don’t care” (GCCAP_dontcare) as transmit codec selection, meaning that the complete list of coders supported by a product is used when negotiating the coder type to be used, also irrespective of this feature enablement.

Note: This feature is only applicable in 1PCC Global Call mode.

1.4.2 Implementation

The Global Call functions, **gc_SetConfigData( )** and **gc_SetUserInfo( )**, are used to enable or disable the feature at the IPT board-level or IPT network device (channel) level. To control the behavior of the G.723.1 bit rate for the entire IPT board, use the **gc_SetConfigData( )** with a Target Type GCTGT_CCLIB_NETIF and the board-level device handle for Target ID. Use **gc_SetUserInfo( )** with the IPT network device (channel) to control the behavior of the G.723.1 bit rate for all calls on a given channel. In both cases the following applies:

The set ID parameter IPSET_CONFIG is used to configure general IP parameters. For this feature, this set ID used a new parameter ID. The pre-existing generic parameter values, IP_DISABLE and IP_ENABLE, are used to disable or enable this feature.

The following table shows the new parameter ID in the IPSET_CONFIG parameter set.

Parameter ID	Data Type & Size	Description	SIP/H.323
IPPARM_SEND_G723_UNSPECIFIED_BITRATE	Type: int Size: size of (int)	Specifies not send any bit rate for G.723.1 on outgoing SIP requests or responses. Valid values: IP_ENABLE 1 (Default) IP_DISABLE 0	SIP only (1PCC)

Note: The new parameter ID is only applicable if both 5.3 kbps and 6.3 kbps G.723.1 bit rates are specified in the capability field in IP_CAPABILITY elements with IP_CAP_DIR_LCLTRANSMIT direction. Otherwise, the IPPARM_SEND_G723_UNSPECIFIED_BITRATE parameter has no effect.

When controlling the new feature behavior at the board-level basis, the **gc_SetConfigData( )** function should be used with the following arguments:

target_type
The target object type. Use GCTGT_CCLIB_NETIF.

cclib_target_id
Indicates the board-level device.

GC_PARM_BLKP target_datap
Pointer to the data from target object. Use set_ID = IPSET_CONFIG and the new parm_ID = IPPARM_SEND_G723_UNSPECIFIED_BITRATE.

This feature is enabled by default, which eliminates the bit rate parameter in SDP bodies in SIP messages from HMP when both bit rates are specified in IP_CAPABILITY structures with IP_CAP_DIR_LCLTRANSMIT as direction. The application has the option to disable this feature using the IP_DISABLE value for IPPARM_SEND_G723_UNSPECIFIED_BITRATE if desired to maintain backward compatibility.

Note: It is recommended that whenever the application desires to specify one G.723.1 bit rate as part of the SIP request, that only one G.723.1 bit rate is passed to the HMP software in a capability field of an IP_CAPABILITY element with IP_CAP_DIR_LCLTRANSMIT as direction. In that particular case, the new parameter ID IPPARM_SEND_G723_UNSPECIFIED_BITRATE is not applicable.

1.4.3 Examples

Since this feature is enabled by default, this example is intended to help customers revert to the previous behavior by disabling the feature if so desired. The example is divided in two parts, the first part demonstrates how to disable this feature on a board-level basis, i.e., make the HMP software behave prior to this feature implementation.

```
char *boardDeviceName = "N_iptB0:P_IP";  // example of board name for gc_openex()
LINEDEV boarddevh = 0;                   // IP board-level line device filled by gc_openex()

INT32 processEvtHandler()
{
GC_PARM_BLK *parmblkp = NULL;
long request_id = 0;

      switch (evtType)
      {
      ...
      case GCEV_OPENEX:
                // board case

                gc_util_insert_parm_val(&parmblkp, IPSET_CONFIG,
                          IPPARM_SEND_G723_UNSPECIFIED_BITRATE, IP_DISABLE);

                if (gc_SetConfigData(GCTGT_CCLIB_NETIF, boarddevh, parmblkp, 0,
                          GCUPDATE_IMMEDIATE, & request_id, EV_ASYNC) != GC_SUCCESS)
                {
                           // Process error
```

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

This second part completes the example by showing how to include both G.723.1 audio bit rates in any outgoing SIP messages with SDP body on a given channel.

```
if (ipcap.capability != GCCAP_DATA_t38UDPFax) {
          ipcap.type = GCCAPTYPE_AUDIO;
          ipcap.direction = IP_CAP_DIR_LCLTRANSMIT;
          ipcap.extra.audio.frames_per_pkt = GCCAP_AUDIO_g7231_6_3k;
          ipcap.extra.audio.VAD = vad;
          /* append the GC_PARM_BLK with the respective TX codec */
          gc_util_insert_parm_ref(&parmblkp, GCSET_CHAN_CAPABILITY,
          IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &ipcap);
          ipcap.type = GCCAPTYPE_AUDIO;
          ipcap.direction = IP_CAP_DIR_LCLTRANSMIT;
          ipcap.extra.audio.frames_per_pkt = GCCAP_AUDIO_g7231_5_3k;
          ipcap.extra.audio.VAD = vad;
          /* append the GC_PARM_BLK with the respective TX codec */
          gc_util_insert_parm_ref(&parmblkp, GCSET_CHAN_CAPABILITY,
          IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &ipcap);


          if (gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev, parmblkp,
                                          scope) != GC_SUCCESS)
          {
                //Process error
          }
}
```

1.5 Overlap-Receive Support for Limited SIP-I Interworking Scenarios

Service Update 240 provides SIP-I interworking capability by providing a method for handling overlap-receive SIP calls, where called-party addressing is supplied in multiple INVITEs but needs to be propagated to the application as standard en bloc signaling calls.

This specific SIP-I interworking scenario is for an overlap signaling call originated at a remote ISUP exchange containing partial DNIS addressing in the called-party number field of a IAM and SAM(s) which are propagated to HMP as embedded MIME bodies in multiple SIP INVITE requests as part of a SIP transaction. In this scenario, each INVITE within the transaction contains an ISUP IAM and zero or more SAM messages embedded in a MIME body. IAM/SAM messages would carry partial called-party addressing until a final INVITE contains the complete called-party addressing within the ISUP IAM and optional SAM messages.

With this feature, HMP in 1PCC mode handles a SIP-I overlap-receive scenario by parsing the ISUP IAM and optional SAM MIME bodies in each INVITE and determining if the called-party address is incomplete. If it is incomplete, HMP automatically rejects it with a 484 Address Incomplete response. The processing continues until the called-party address is complete, at which point the application is presented with the incoming call as if it were a standard en bloc signaling INVITE.

1.5.1 Feature Overview

This feature is concerned with ITU-T Q.763 IAM and SAM messages, in particular with the called-party number field in them, containing the DNIS. When this feature is implemented, HMP, acting as a SIP UAS, can support specific SIP-I overlap-receive scenarios where an interworking unit (gateway) at the far end embeds the IAM/SAM message in a multipart MIME body as part of the INVITE requests, in addition to the SDP MIME part. Each INVITE carries embedded ISUP IAM and optional SAM MIME bodies with partial called-party addressing (DNIS). In these scenarios, the first INVITE normally carries an ISUP IAM from the originating exchange, and subsequent INVITEs carry ISUP SAMs belonging to the same call transaction. The ISUP IAM contains an initial called-party addressing and a number of ISUP SAMs will contain additional called-party addressing until the complete called-party address signaling is provided.

Multiple INVITEs containing ISUP IAM and zero or more SAM messages, each one containing the original addressing plus additional addressing, are received by HMP until the called-party addressing is complete, at which point an end of signaling (stop digit) appears at the end of the called-party number field, indicating a complete address. This digit uniquely identifies the last INVITE in the sequence.

When this feature is enabled, HMP parses the MIME body in each incoming INVITE request to determine if ISUP IAM or SAM messages in ITU-T Q.763 format are present. If so, the called-party number field is parsed looking for an end of signaling (stop digit “F”) presence. If the IAM or SAM message does not contain the stop digit, then the INVITE is automatically rejected by HMP with a 484 Address Incomplete response with no application interaction. No further processing by HMP is done on this INVITE, nor is the application made aware of the automatic 484 Address Incomplete response. If the stop digit (“F”) is present, then the INVITE is presented to the application as a standard GCEV_OFFERED event with the complete en bloc DNIS address, which will be contained in the INVITE header fields.

1.5.2 Enabling MIME-based Overlap-Receive

The application enables MIME-based Overlap-Receive on a board basis using **gc_SetConfigData( )** function. The IPSET_CONFIG set ID is used in conjunction with a new parameter ID, IPPARM_MIME_OVERLAP_RECEIVE, defined as follows:

Parameter ID	Data Type	Description	SIP/ H.323
IPPARM_MIME_OVERLAP_ RECEIVE	Type: int Size: size of (int)	Enables the overlap receive feature of SIP-I based on embedded MIME ISUP messages. Possible values are: • GCPV_ENABLE – ISUP IAM and optional SAM messages embedded in a MIME body in the incoming INVITE message are parsed to check if the "F" stop digit is present. If not present, the call will be rejected with a 484 Address Incomplete response. • GCPV_DISABLE – ISUP MIME bodies will not be parsed. This is the default behavior.	SIP only

Once the feature is enabled at the board-level basis, all IPT network devices (channels) are able to parse MIME bodies carrying ISUP IAM and SAM messages in ITU-T Q.763 format containing overlap called-party number fields and automatically reject those INVITEs within the transaction that do not contain complete addressing. Once the complete address is received, HMP presents the incoming call and its complete called-party addressing as a standard GCEV_OFFERED event to a channel. The channel can handle the incoming event as it would normally do with any other incoming call. The called-party addressing signaling is expected to be available in the relevant header fields within the INVITE which will be made available to the channel as a standard DNIS number.

Notes:1. HMP requires that ISUP messages embedded as MIME bodies in the incoming INVITEs are of IAM or SAM type and adhere to the ITU-T Q.763 specification for the feature to be able to parse overlap-receive signaling.

2. ISUP SAM messages are not always required to carry additional called-party signaling, and a single IAM message may be sufficient to carry complete en bloc called-party addressing.

3. It is possible for one INVITE to carry an ISUP IAM and one or more SAMs, each containing partial called-party number fields.

Example

The following code shows how to enable this feature:

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

/*
```

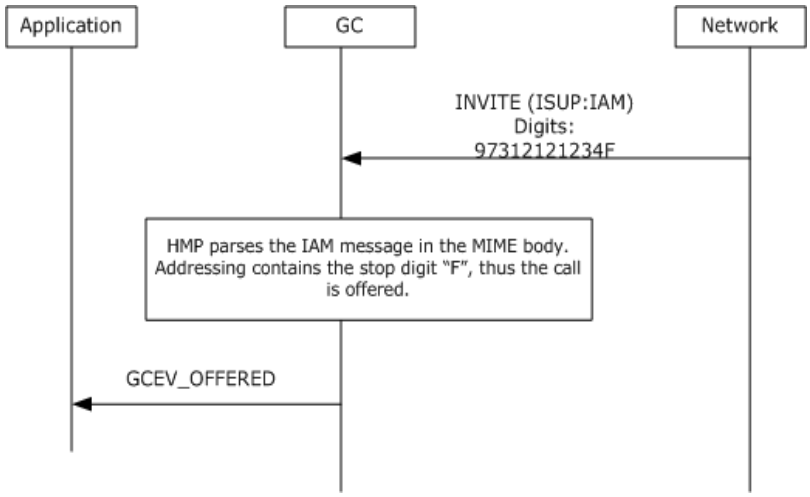
```
* Assume the following has been done:
* 1. Open board device iptB1 and save handle as boardDev which will be
* used in gc_SetConfigData()
*/

int enable_overlap()
{
        GC_PARM_BLKP parmblkp = NULL;
        long request_id = 0;
        gc_util_insert_parm_val(&parmblkp,
               IPSET_CONFIG,
               IPPARM_MIME_OVERLAP_RECEIVE,
               sizeof(int),
               GCPV_ENABLE);

         if(gc_SetConfigData(GCTGT_CCLIB_NETIF, boardDev, parmblkp, 0, /*timeout*/,
             GCUPDATE_IMMEDIATE, &request_id, EV_ASYNC) != GC_SUCCESS)
         {
              // handle error…
         }

         return (0);
}
```

1.5.3 Use Cases

The overlap-receive feature involves one or more INVITE messages sent by the remote UAC. Each INVITE may contain initial and additional called-party addressing in embedded ISUP IAM and optional SAM MIME bodies within the request. The ISUP IAM and optional SAM bodies are parsed seeking for the called-party number field. Each INVITE is treated independent of any previous INVITE in the transaction until the last INVITE with complete addressing is received. The following figures demonstrate typical use cases for this feature.

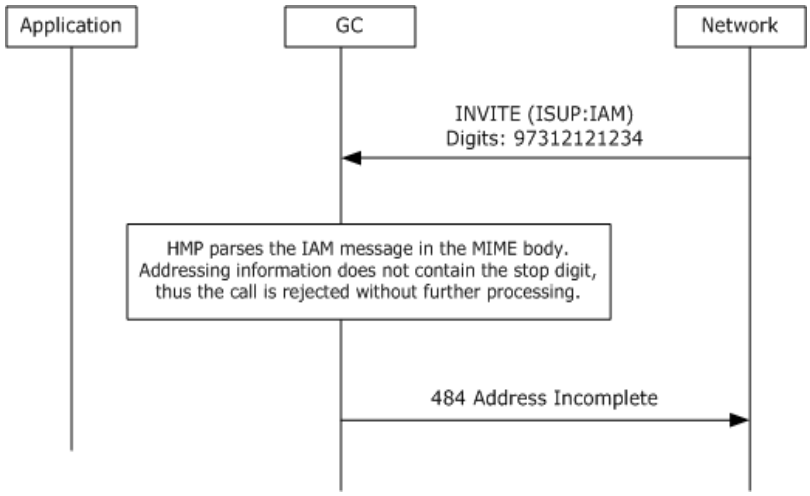
INVITE with Complete Address in IAM

The incoming INVITE contains an IAM message with the complete address (contains the stop digit “F”).



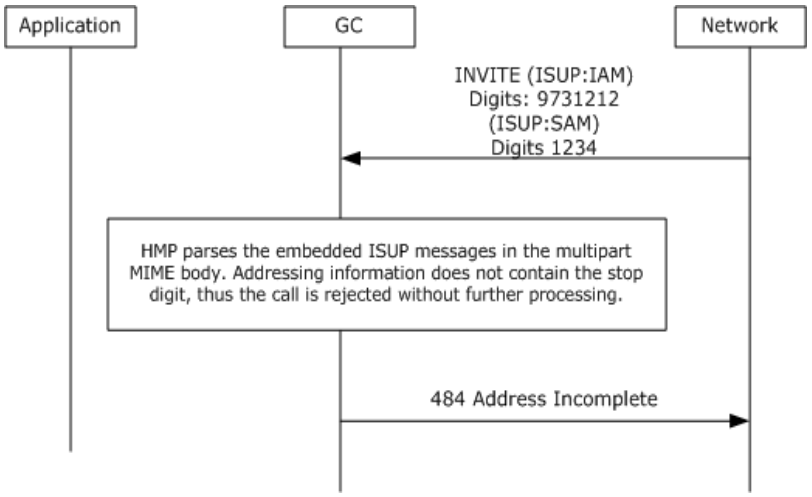
INVITE with Incomplete Address in IAM

The incoming INVITE contains an IAM message with incomplete address (does not contain the stop digit).



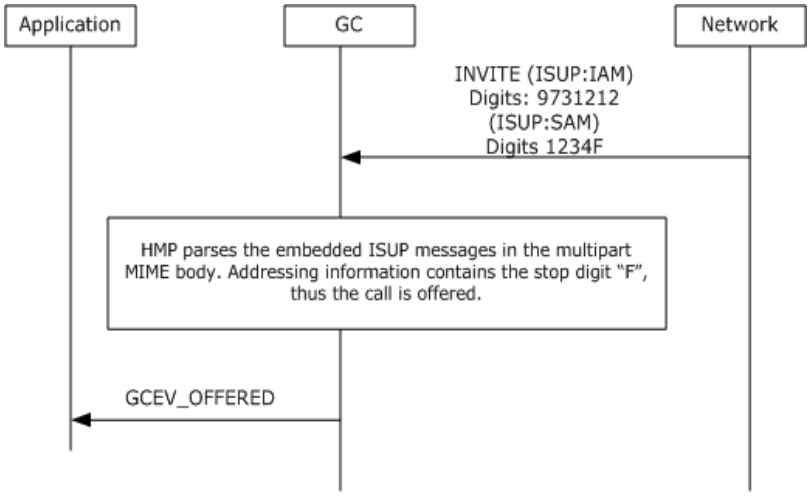
INVITE with Incomplete Address in IAM and SAM

The incoming INVITE contains an IAM and a SAM, and neither message contains the complete address (neither one contains the stop digit).



INVITE with Complete Address in IAM and SAM

The incoming INVITE contains both IAM and SAM, and the messages contain the complete address (contains the stop digit “F”).



For additional information about MIME bodies, refer to Section 4.10.1, SIP MIME Overview, in the Dialogic® Global Call IP Technology Guide.

1.6 Processing Multiple 18x Provisional Responses

Service Update 240 introduces a method for obtaining subsequent provisional 18x SIP responses using the GCEV_EXTENSION event. When this feature is enabled, the first incoming 18x response will generate a GCEV_ALERTING as expected; however, all subsequent 18x responses will be sent to application by the GCEV_EXTENSION event. The block parameter associated with the event will contain SIP response headers in the same block format as in the GCEV_ALERTING event for the first provisional response received on the channel for the same transaction.

The following 18x provisional responses are supported:

- 180 Ringing
- 181 Call is Being Forwarded
- 182 Queued
- 183 Session Progress

1.6.1 Feature Implementation

A new bitmask value, EXTENSIONEVT_SIP_18x_RESPONSE, is added to the existing IPSET_EXTENSIONEVT_MSK parameter set for feature enablement on a board-level basis using the **gc_SetConfigData( )** function with the GCTGT_CCLIB_NETIF target_type and the board-level device handle as the target ID. Once enabled, all IPT network devices (channels) will have the ability to be notified of multiple18x provisional responses from the UAS.

Since the GCEV_EXTENSION has several variances for IP, for example, it can be used to retrieve call-related information, get notification of underlying protocol connection or disconnection, or get notification of media streaming initiation and termination, a new extension ID, IPEXTID_RECEIVED_18x_RESPONSE, is introduced so a GCEV_EXTENSION event for this feature can be differentiated from another feature.

Once the GCEV_EXTENSION event is received, the application can easily find the reason by looking at the ext_id which is part of the EXTENSIONEVTBLK embedded in the event data pointer.

Furthermore, if the ext_id matches IPEXTID_RECEIVED_18x_RESPONSE, the 18x provisional response headers can be found in the same format as in the GCEV_ALERTING event for the first provisional response received on the channel in the same transaction. That is, the parmblkp associated with the GCEV_EXTENSION event contains the following mandatory set_ID and parm_ID:

- set_ID = IPSET_SIP_RESPONSE_CODE
- parm_ID = IPPARM_RECEIVED_RESPONSE_STATUS_CODE
- value = 3-digit integer representing the Status-Code from the response's Status-Line

If Reason-Phrase retrieval from 182 and 183 Provisional Responses has been enabled, then the parmblkp associated with the GCEV_EXTENSION event contains set_ID and parm_ID:

- set_ID = IPSET_MSG_INFO
- parm_ID = IPPARM_SIP_HDR
- value = NULL-terminated string which begins with the string “Reason-Phrase"

Processing Responses

The following behavior occurs within the application when the feature is enabled:

The first incoming 18x response will generate a GCEV_ALERTING as expected, or if a 183 was received and the application enabled support for Ingress 183 Session Progress Informational Response Containing SDP, a GCEV_PROGRESSING is received instead.

The GCEV_EXTENSION event indicates the receipt of any subsequent 18x responses within a transaction. The METAEVENT structure associated with it contains:

- EXTENSIONEVTBLK.ext_id = IPEXTID_RECEIVED_18x_RESPONSE
- EXTENSIONEVTBLK.parmblk will have at least one (mandatory) entry with the following (reused from GCEV_ALERTING):
  set_ID = IPSET_SIP_RESPONSE_CODE
  parm_ID = IPPARM_RECEIVED_RESPONSE_STATUS_CODE
  value = 3-digit integer representing the Status-Code from Status-Line of the received provisional response

If Reason-Phrase retrieval from 182 and 183 Provisional Responses is enabled, the EXTENSIONEVTBLK.parmblk will contain the following additional entry:

- set_ID = IPSET_MSG_INFO
- parm_ID = IPPARM_SIP_HDR
- value = NULL-terminated string which begins with the string “Reason-Phrase” and contains the equivalent header from the response’s Status-Line

Note: This feature is activated on the board level; however, the GCEV_EXTENSION event is IPT network device (channel) specific.

1.6.2 Enabling/Disabling GCEV_EXTENSION

Enabling and disabling unsolicited GCEV_EXTENSION notification events is done by manipulating the event mask using the **gc_SetConfigData( )** function as described in Section 4.6.1, Enabling and Disabling Unsolicited Notification Events, of the *Dialogic® Global Call IP Technology Guide.*

The following example shows how to set the EXTENSIONEVT_SIP_18X_RESPONSE mask:

```
char *boardDeviceName = "N_iptB0:P_IP"; // example of board name for gc_openex()
LINEDEV boardDevice = 0; // IP channel board-level line device filled by gc_openex()

static void evt_hdlr()
{
...
       case GCEV_OPENEX:
       ...
       // board case
       {

              GC_PARM_BLK * pparm = NULL;
              long req_id;

              gc_util_insert_parm_val(pparm,
                     IPSET_EXTENSIONEVT_MSK,
                     GCACT_ADDMSK,
                     GC_VALUE_LONG,
                     EXTENSIONEVT_SIP_18X_RESPONSE);

              gc_setConfigData(GCTGT_CCLIB_NETIF,
                      boardDevice,
                      pparm,
                      0,
                      GCUPDATE_IMMEDIATE,
                      &req_id,
                      EV_ASYNC);
        }
...
}
```

1.6.3 Retrieving 18x Code from GCEV_EXTENSION

The following example demonstrates how to retrieve the Status-Code and Reason-Phrase headers (if available) from an18x response mapped to a GCEV_EXTENSION unsolicited event once the feature is enabled. Refer to Section 4.4.3.2, Retrieving Reason-Phrase from 182 and 183 Provisional Responses, in the *Dialogic® Global Call IP Technology Guide* for information about setting up the application to receive Reason-Phrase header from these SIP responses.

```
case GCEV_EXTENSION:
GC_PARM_BLKP pParmBlock;
EXTENSIONEVTBLK *pextensionBlk;
GC_PARM_DATAP l_pParmData;
pextensionBlk = (EXTENSIONEVTBLK *)(m_pMetaEvent->extevtdatap);
pParmBlock = (&(pextensionBlk->parmblk));

if(pextensionBlk->ext_id== IPEXTID_RECEIVED_18X_RESPONSE)
{
      while ((l_pParm = gc_util_next_parm(pParmBlock, l_pParm)) != 0)
      {
             int l_mtype= (int)(*( l_pParm->value_buf));
             switch (l_pParm->set_ID)
             {
             case IPSET_SIP_RESPONSE_CODE:
                    if (l_pParm->parm_ID == IPPARM_RECEIVED_RESPONSE_STATUS_CODE)
                    {

                          if (l_pParmData->value_size != 0)
                          {
                                 int code_18x = *(int *) l_pParm ->value_buf;
                                 //...
                          }
```

```
                        }
                        break;
                case IPSET_SIP_MSGINFO:
                        if (l_pParm->parm_ID == IPPARM_SIP_HDR)
                        {
                              if (l_pParmData->value_size != 0)
                              {
                                    char siphdr_18x[IP_SIP_HDR_MAXLEN];

                                    strncpy(siphdr_18x,(char*)parmp->value_buf,parmp->value_size);

                                    siphdr_18x[parmp->value_size]='\0';
                                    //...
                               }
                        }
                        break;
                default:
                        //... warning no other type allowed
                        break;
                }
          }
}
else
{
          // other ext_id
}
break;

... other GCEV_ case
```

1.7 TLS and SRTP Channel Support Increase

With Service Update 240, Dialogic® HMP Software now supports 500 Transport Layer Security (TLS) channels and 500 Secure Real Time Protocol (SRTP) channels. Previously, the software was limited to 250 channels of each.

1.8 33 Frames Per Packet Support (AMR)

Service Update 240 increases the maximum frames per packet from 10 to 33 for AMR.

Note: Voice activity detection (VAD) is disabled by firmware when frames per packet is greater than 10. This cannot be changed by the application.

Refer to Section 3.2.2, “Dialogic® Host Media Processing Software for Linux Configuration Guide”, on page 109 for documentation updates due to this feature.

1.9 Registering Authentication Data without Realm String

Service Update 240 introduces a method for registering authentication data without using realm string.

Currently, the realm element of the authentication quadruplet must contain a non-empty string. This feature supports realm and identity as an empty string, so that identity authentication is verified using the same identity or username as the registrar server.

If the realm value is empty, the application uses the identity element to match with the identity provided by the server.

["",identity, username, password]

If realm and identity are empty, the application can use the username. The username must be the same username in the string identity provided by the registrar server.

["","", username, password]

For example, if the registrar server provides [bob@10.10.20.25] as identity, then "bob" must be configured as the username for the registrar server and the authentication quadruplet.

For more information about SIP Digest Authentication, refer to the *Dialogic® Global Call IP Technology Guide*.

1.10 Handling non-2xx Responses to T.38 Switch

Service Update 240 introduces 1PCC Global Call support for RFC3261 compliance for non-2xx responses to re-INVITE requests to switch to or from audio to T.38 fax and back.

1.10.1 Feature Description

Currently, when a Global Call SIP application initiates a media type switch from/to audio or to/from fax within a dialog with a re-INVITE request, the existing media session and dialog are terminated if the request is rejected by the UAS with any non-2xx response. RFC3261 clearly requires that the UAC keep the exiting session in a dialog alive as though the re-INVITE never occurred.

With this feature, the existing media session within the dialog remains active upon a switching request from one media type to another (fax to audio or audio to fax).

This feature is enabled by default when the application calls the **gc_ReqModifyCall( )** function or the **gc_Extension( )** function with the codec switch value. On failure to switch, the application will receive the failure events, GCEV_REQ_MODIFY_REJ/GCEV_REQ_MODIFY_FAIL and the GCEV_EXTENSION event with parm ID set to IPPARM_REJECT for set ID IPSET_SWITCH_CODEC respectively. The existing media session will be reestablished underneath, and the requested local media information will contain the stored (existing) media information.

Because this feature is limited to “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.

The operating mode for set ID/parm ID pair IPSET_CONFIG/IPPARM_OPERATING_MODE must be set to either of the following:

- IP_T38_MANUAL_MODE
- IP_T38_MANUAL_MODIFY_MODE

For additional information, refer to the documentation updates for Chapter 3. IP Call Scenarios and Chapter 4. IP Specific Operations in the Dialogic Global Call IP Technology Guide.

1.10.2 Manual Mode Example

This example demonstrates “Manual” mode when the switch from T.38 fax to audio is unsuccessful.

```
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;
      GC_INFO    t_info;

      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_REJECT:
```

```
                gc_ResultInfo(&metaEvent,&t_info);
                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);
                break;
                case IPPARM_READY:
                /* Ready to send and receive audio */
                  gc_Listen();
                break;
           }
}
```

This example demonstrates “Manual” mode when the switch from audio to fax is unsuccessful.

```
INT32 processEvtHandler( )
{
METAEVENT metaEvent;
GC_PARM_BLK *parmblkp = NULL;
IP_CONNECT ipConnect;
GC_INFO    t_info;

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_REJECT:

                gc_ResultInfo(&metaEvent,&t_info);
```

```
                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_Listen();
                /* IPT to IPM*/
                break;
            case IPPARM_READY:
            /* Ready to send and receive fax */
            fx_sendfax();
            break;
        }
        break;
}
```

1.10.3 Manual Modify Mode Examples

This example demonstrates “Manual” modify mode when the switch from T.38 fax to audio is unsuccessful.

```
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 */
      if( gc_util_insert_parm_ref(&parmblkp,GCSET_CHAN_CAPABILITY, IPPARM_LOCAL_CAPABILITY,
          sizeof(IP_CAPABILITY), &ipcap) != GC_SUCCESS )
      {
       //error
      }
      gc_ReqModifyCall (GCTGT_GCLIB_CRN,pline->crn, parmblkp, 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_MODIFY_CALL_ACK:
        // switch complete
          gc_Listen();
        break;

        case GCEV_MODIFY_CALL_REJ:
```

```
        case GCEV_MODIFY_CALL_FAIL:
        gc_util_insert_parm_ref(&parmblkp, IPSET_FOIP, IPPARM_T38_CONNECT,
                                (sizeof(IP_CONNECT)), (void *)(&ipConnect));
        break;
      }
```

This example demonstrates “Manual” modify mode when the switch from audio to fax is unsuccessful.

```
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);

     if( gc_util_insert_parm_ref(&parmblkp, GCSET_CHAN_CAPABILITY,
         IPPARM_LOCAL_CAPABILITY, sizeof(IP_CAPABILITY), &ipcap) != GC_SUCCESS )
     {
     //error
     }
     gc_ReqModifyCall (GCTGT_GCLIB_CRN,pline->crn, parmblkp, EV_ASYNC);
     gc_util_delete_parm_blk(parmblkp);
     break;

     case GCEV_MODIFY_CALL_ACK:
         // Switch Complete
         fx_sendfax();
         break;

     case GCEV_MODIFY_CALL_REJ:
     case GCEV_MODIFY_CALL_FAIL:

          /* received extension event for media readiness */
          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_Listen();
          /* IPT to IPM*/
                break;
}
```

1.11 MIME Insertion in Outgoing ACK

Service Update 240 introduces a method for embedding a MIME body in an outgoing SIP ACK request message for 4xx/5xx/6xx response messages that terminate certain transactions.

1.11.1 Feature Description

Some interworking SIP configurations use embedded MIME bodies in messages to pass call state information to a non-SIP destination network. Although Dialogic® HMP Software supports the encapsulation of MIME bodies in many request messages out of HMP, this is not the case for ACK requests.

In particular, when an outbound call out of HMP (UAC) cannot be completed by the remote UAS, it will send a rejection/ message using the appropriate 4xx, 5xx, or 6xx SIP final response message.

Currently, once a 4xx/5xx/6xx response is received, the stack automatically generates the ACK request, terminates the transaction, and informs the application of the failure with a GCEV_DISCONNECTED event. Since the ACK is automatically generated without application intervention, MIME body encapsulation is not possible.

This feature allows the application to create a MIME body ahead of time so that it is then available for the IP cclib to add it to outgoing ACK requests that are a result of certain outbound call rejections.

Once the feature is enabled at the IPT network device (channel), and the application builds the desired MIME body ahead of time, the MIME body will be added ACK messages out of HMP that are generated as a response to any 4xx/5xx/6xx SIP final response messages in most transaction scenarios.

An exception is the 487 “Request Terminated” message, generated by the server (UAS) as a final response to a CANCEL to a previous INVITE out of HMP. In this case, the ACK message for this 487 will not contain any MIME body irrespective of feature enablement. The **gc_DropCall( )** that generated the ACK will internally disable MIME body encapsulation to the subsequent ACK to this 487.

Note: A 487 message generated by the server (UAC) resulting from a transaction timer expiration, as per an “Expires” header in the outgoing INVITE out of HMP, will be handled like most 4xx/5xx/6xx final response scenarios. The ACK message sent in this case will contain a MIME body pre-built by the application as part of this feature.

Disclaimers

- The feature is intended for use with proxy or general UAS that are able to handle encapsulated bodies in ACK messages. In particular, IETF RFC 3261 SIP stipulates that the placement of bodies in ACK for non-2xx is NOT RECOMMENDED since they cannot be rejected if the body is not understood.
- If bodies are to be inserted, IETF RFC 3261 SIP RECOMMENDS that they be the same as they appeared in the original INVITE. This feature does not restrict the building of any type of valid MIME bodies, and the application is responsible for MIME content generation and applicability.
- Although this feature is exemplified in a SIP <-> ISDN User Part (ISUP) interworking, known as SIP-I, it does not imply SIP-I compliance. The feature is very specific to limited outbound scenarios.

1.11.2 Enabling MIME Insertion

To implement this feature, a new set ID is introduced. This set ID is defined as:

#define IPSET_MIME_ACK_TO_REJECTION BASE_SETID+42

It may be used with the **gc_SetUserInfo( )** function, along with target_type GCTGT_GCLIB_CHAN, and duration set to GC_SINGLECALL.

The method of building the MIME body using IPSET_MIME_ACK_TO_REJECTION is the same as the method used for building the two-level GC_PARM_BLK data structures as documented in the *Dialogic® Global Call IP Technology Guide* for IPSET_MIME or IPSET_MIME_200OKTOBYE.

Note: To create MIME bodies requires the manipulation of MIME bodies to be enabled at the IP virtual board level, prior to a calling the **gc_Start( )** function, and using the mask sip_msginfo_mask = IP_SIP_MSGINFO_ENABLE | IP_SIP_MIME_ENABLE.

1.11.3 Setting the MIME Body for the ACK Message

After the feature is enabled, and prior to making an outbound call, the application creates the MIME body containing the desired data to be sent along with ACK responses to final 4xx/5xx/6xx responses for those transactions as defined in the Feature Description section.

The following table shows the parameter IDs in the IPSET_MIME_ACK_TO_REJECTION set ID:

Parameter ID	Data Type & Size	Description	SIP/ H.323
IPPARM_MIME_PART	Type: pointer to GC_PARM_BLK Size: 4 bytes	Required parameter. Used to set or get SIP message MIME part(s). Parameter value is a pointer to a GC_PARM_BLK structure that contains a list of pointers to one or more GC_PARM_BLK structures that contain MIME message parts.	SIP only
IPPARM_ MIME_PART_BODY	Type: char * Size: 4 bytes	Required parameter. Used to copy MIME part body between application and Global Call space. Parameter value is a pointer to a MIME part body.	SIP only
IPPARM_ MIME_PART_BODY_SIZE	Type: Unsigned int Size: 4 bytes	Required parameter. Used to indicate the actual size of the MIME part body, not including MIME part headers.	SIP only
† For parameters with data of type String, the length in a GC_PARM_BLK is the length of the data string plus 1.			

Parameter ID	Data Type & Size	Description	SIP/ H.323
IPPARM_ MIME_PART_HEADER	Type: Null-terminated string † Size: max. length = max_parm_data_size (configured at start-up via IPCCLIB_START_DATA)	Optional parameter. Used to contain MIME part header field in format of "field-name: field-value". Field-name can be any string other than "Content-type". Content is not checked by Global Call before insertion into SIP message.	SIP only
IPPARM_ MIME_PART_TYPE	Type: Null-terminated string † Size: max. length = max_parm_data_size (configured at start-up via IPCCLIB_START_DATA)	Required parameter. Used to contain name and value of the MIME part content type field. String must begin with the field name "Content-Type:".	SIP only
† For parameters with data of type String, the length in a GC_PARM_BLK is the length of the data string plus 1.			

Notes:1. The application is responsible for building the appropriate MIME bodies. IP cclib does not make any attempt to parse the raw data in the MIME, and encapsulates it in the outgoing ACK as built.

2. IPSET_MIME_ACK_TO_REJECTION is supported only on a channel basis specifically for the GCTGT_GCLIB_CHAN target type. The MIME message set is not valid after the call is cleared. For the next outbound call, the application must call the **gc_SetUserInfo( )** function to reset the MIME body.

For more information about creating MIME messages, refer to the *Dialogic® Global Call IP Technology Guide*, section 4.10.

1.11.4 Possible Use Case and Code Example

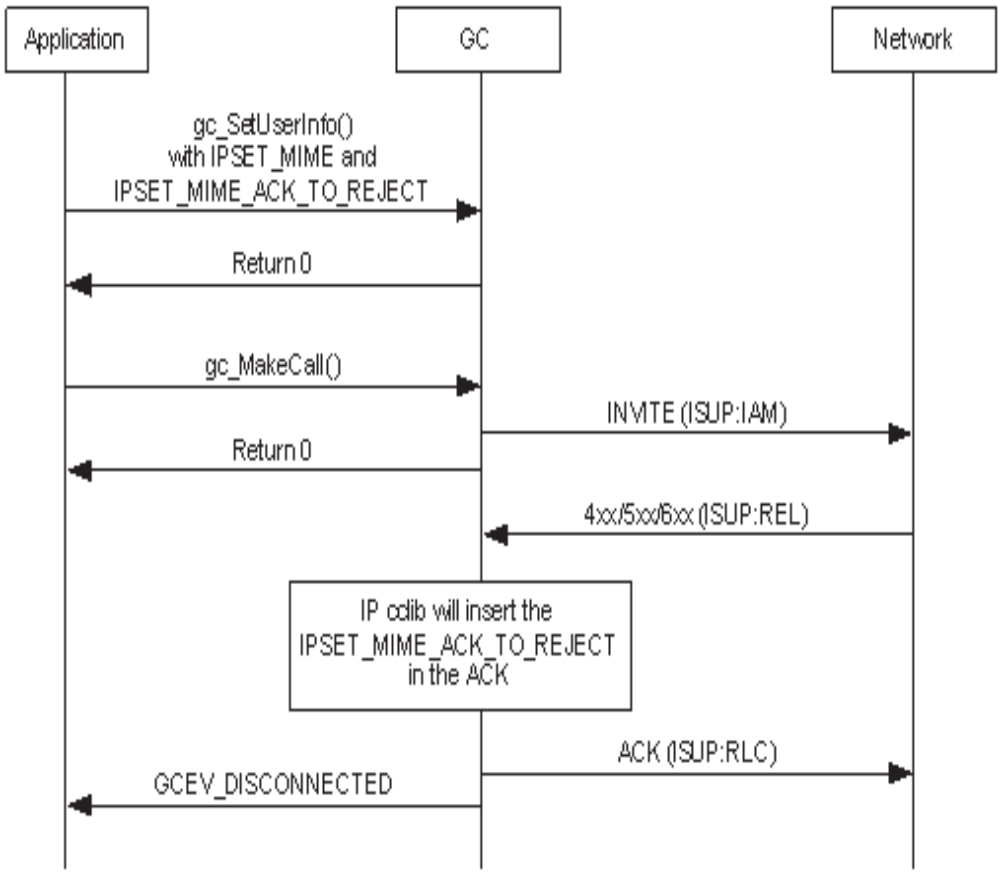
The following code examples illustrate how to enable this feature and send embedded MIME bodies in outgoing ACK response messages. The examples assume that the MIME body manipulation mask was enabled using the INIT_IP_VIRTBOARD data structure.

The following diagram shows a very specific example of encapsulating MIME bodies in two different request messages out of HMP, one of them is specific to this feature. This use case scenario represents a possible SIP <-> ISUP interworking scenario.

Notes:1. This is for a specific use-case scenario. It is not intended to imply that this is the only SIP interworking use case where this feature may be used.

2. The following diagram and code shows how to send a possible ISUP:IAM body as part of an outgoing INVITE out of HMP. This functionality is already available in HMP as part of the IPSET_MIME set ID.

3. An example ISUP:RLC body is sent automatically (via this feature) as part of an outgoing ACK out of HMP from this 4xx/5xx/6xx final rejection response from UAS.



Two different MIME bodies are to be created, one for the INVITE request and the other one for the ACK request (this feature) out of HMP. The examples show the following two possible ways of building the two MIME bodies:

- Single call to **gc_SetUserInfo( )**
- Two separate **gc_SetUserInfo( )** calls

Note: The MIME bodies in this example are purely for the sake of a complete example, and should not be relied on to contain valid ISUP messages.

Single call to **gc_SetUserInfo( )**

```
void  setMIMEInfo(int index)
{
    char str[MAX_STRING_SIZE];
    int frc;

    /* The following variable is used for the MAIN GC_PARM_BLK that
     * will contain the different MIME blocks */
    GC_PARM_BLK *pParmBlockMain = NULL;

    /* The following variables are used for the IAM MIME */
    GC_PARM_BLK *pParmBlockIAM_B = NULL;
```

```
    char *pBodyTypeIAM = "Content-Type: application/ISUP; version = itu-t92+";
    char *pBodyIAM = "01 00 49 00 00 03 02 00 07 04 10 00 33 63 21\r\n43 00 00 03 06 0d 03 80 90
a2 07 03 10 03 63\r\n53 00 10 0a 07 03 10 27 80 88 03 00 00 89 8b\r\n0e 95 1e 1e 1e 06 26 05 0d
f5 01 06 10 04 00";
    char *pPartHeaderIAM1 = "Content-Disposition: signal ;handling=optional";
    char *pPartHeaderIAM2 = "Content-User: Dialogic ;type=demo";

    /* The following variables are used for the RLC MIME */
    GC_PARM_BLK *pParmBlockRLC_B = NULL;

    char *pBodyTypeRLC = "Content-Type: application/ISUP; version = itu-t92+";
    char *pBodyRLC = "01 00 49 00 00 03 02 00 07 04 10 00 33 63 21\r\n43 00 00 03 06 0d 03 80 90
a2 07 03 10 03 63";
    char *pPartHeaderRLC1 = "Content-Disposition: signal ;handling=optional";
    char *pPartHeaderRLC2 = "Content-User: Dialogic ;type=demo";


    /* Set the IAM MIME */
    /* Add 1 to strlen for the NULL termination character */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_TYPE,
                               (unsigned long)(strlen(pBodyTypeIAM) + 1),
                               pBodyTypeIAM);

    /* Insert Body Size */
    gc_util_insert_parm_val(&pParmBlockIAM_B,
                            IPSET_MIME,
                            IPPARM_MIME_PART_BODY_SIZE,
                            sizeof(unsigned long),
                            strlen(pBodyIAM));

    /* Insert MIME part Body Pointer */
    gc_util_insert_parm_val(&pParmBlockIAM_B,
                            IPSET_MIME,
                            IPPARM_MIME_PART_BODY,
                            sizeof(unsigned long),
                            (unsigned long)pBodyIAM);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderIAM1) + 1),
                               pPartHeaderIAM1);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderIAM2) + 1),
                               pPartHeaderIAM2);

    /* Insert parm block B pointer to Main parm block */
    gc_util_insert_parm_val(&pParmBlockMain,
                            IPSET_MIME,
                            IPPARM_MIME_PART,
                            sizeof(unsigned long),
                            (unsigned long)pParmBlockIAM_B);

    /* Set the RLC MIME */
    /* Add 1 to strlen for the NULL termination character */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_TYPE,
                               (unsigned long)(strlen(pBodyTypeRLC) + 1),
                               pBodyTypeRLC);
```

```
    /* Insert Body Size */
    gc_util_insert_parm_val(&pParmBlockRLC_B,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART_BODY_SIZE,
                            sizeof(unsigned long),
                            strlen(pBodyRLC));

    /* Insert MIME part Body Pointer */
    gc_util_insert_parm_val(&pParmBlockRLC_B,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART_BODY,
                            sizeof(unsigned long),
                            (unsigned long)pBodyRLC);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderRLC1) + 1),
                               pPartHeaderRLC1);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderRLC2) + 1),
                               pPartHeaderRLC2);

    /* Insert parm block B pointer to MAIN parm block */
    gc_util_insert_parm_val(&pParmBlockMain,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART,
                            sizeof(unsigned long),
                            (unsigned long)pParmBlockRLC_B);

    frc = gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev,pParmBlockMain,GC_SINGLECALL);
    if(GC_SUCCESS != frc)
    {
        sprintf(str, "gc_SetUserInfo failed");
        printandlog(index, GC_APIERR, NULL, str, 0);
    }

    gc_util_delete_parm_blk(pParmBlockIAM_B);
    gc_util_delete_parm_blk(pParmBlockRLC_B);
    gc_util_delete_parm_blk(pParmBlockMain);

}
```

Two separate calls to **gc_SetUserInfo( )**

```
void  setMIMEInfo(int index)
{
    char str[MAX_STRING_SIZE];
    int frc;

    /* The following variable is used for the MAIN GC_PARM_BLK that
     * will contain the different MIME blocks */
    GC_PARM_BLK *pParmBlockMain = NULL;

    /* The following variables are used for the IAM MIME */
    GC_PARM_BLK *pParmBlockIAM_B = NULL;

    char *pBodyTypeIAM = "Content-Type: application/ISUP; version = itu-t92+";
    char *pBodyIAM = "01 00 49 00 00 03 02 00 07 04 10 00 33 63 21\r\n43 00 00 03 06 0d 03 80 90
a2 07 03 10 03 63\r\n53 00 10 0a 07 03 10 27 80 88 03 00 00 89 8b\r\n0e 95 1e 1e 1e 06 26 05 0d
```

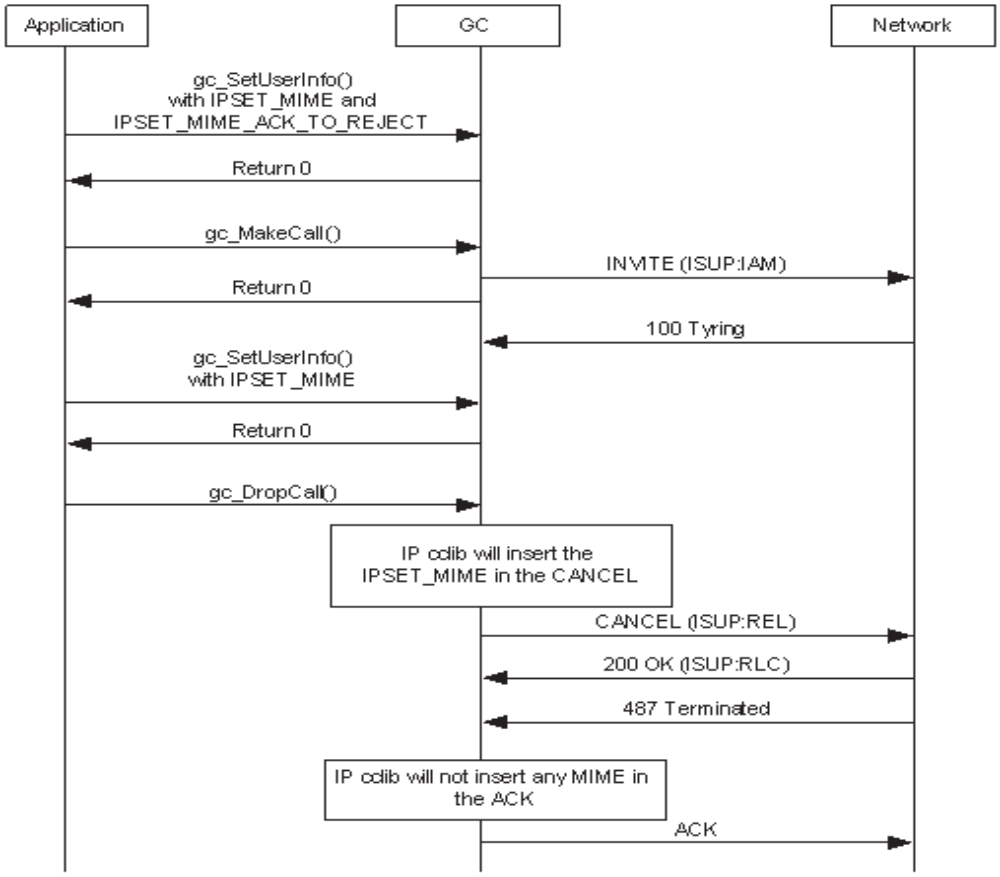
```
f5 01 06 10 04 00";
    char *pPartHeaderIAM1 = "Content-Disposition: signal ;handling=optional";
    char *pPartHeaderIAM2 = "Content-User: Dialogic ;type=demo";

    /* The following variables are used for the RLC MIME */
    GC_PARM_BLK *pParmBlockRLC_B = NULL;
|
    char *pBodyTypeRLC = "Content-Type: application/ISUP; version = itu-t92+";
    char *pBodyRLC = "01 00 49 00 00 03 02 00 07 04 10 00 33 63 21\r\n43 00 00 03 06 0d 03 80 90
a2 07 03 10 03 63";
    char *pPartHeaderRLC1 = "Content-Disposition: signal ;handling=optional";
    char *pPartHeaderRLC2 = "Content-User: Dialogic ;type=demo";

    /* Set the IAM MIME */
    /* Add 1 to strlen for the NULL termination character */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_TYPE,
                               (unsigned long)(strlen(pBodyTypeIAM) + 1),
                               pBodyTypeIAM);

    /* Insert Body Size */
    gc_util_insert_parm_val(&pParmBlockIAM_B,
                            IPSET_MIME,
                            IPPARM_MIME_PART_BODY_SIZE,
                            sizeof(unsigned long),
                            strlen(pBodyIAM));

    /* Insert MIME part Body Pointer */
    gc_util_insert_parm_val(&pParmBlockIAM_B,
                            IPSET_MIME,
                            IPPARM_MIME_PART_BODY,
                            sizeof(unsigned long),
                            (unsigned long)pBodyIAM);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderIAM1) + 1),
                               pPartHeaderIAM1);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockIAM_B,
                               IPSET_MIME,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderIAM2) + 1),
                               pPartHeaderIAM2);

    /* Insert parm block B pointer to Main parm block */
    gc_util_insert_parm_val(&pParmBlockMain,
                            IPSET_MIME,
                            IPPARM_MIME_PART,
                            sizeof(unsigned long),
                            (unsigned long)pParmBlockIAM_B);

    frc = gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev,pParmBlockMain,GC_SINGLECALL);
    if(GC_SUCCESS != frc)
    {
        sprintf(str, "gc_SetUserInfo failed for IAM MIME");
        printandlog(index, GC_APIERR, NULL, str, 0);
    }

    gc_util_delete_parm_blk(pParmBlockIAM_B);
    gc_util_delete_parm_blk(pParmBlockMain);

    pParmBlockMain = NULL;
```

```
    /* Set the RLC MIME */
    /* Add 1 to strlen for the NULL termination character */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_TYPE,
                               (unsigned long)(strlen(pBodyTypeRLC) + 1),
                               pBodyTypeRLC);

    /* Insert Body Size */
    gc_util_insert_parm_val(&pParmBlockRLC_B,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART_BODY_SIZE,
                            sizeof(unsigned long),
                            strlen(pBodyRLC));

    /* Insert MIME part Body Pointer */
    gc_util_insert_parm_val(&pParmBlockRLC_B,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART_BODY,
                            sizeof(unsigned long),
                            (unsigned long)pBodyRLC);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderRLC1) + 1),
                               pPartHeaderRLC1);

    /* Insert other header fields */
    gc_util_insert_parm_ref_ex(&pParmBlockRLC_B,
                               IPSET_MIME_ACK_TO_REJECTION,
                               IPPARM_MIME_PART_HEADER,
                               (unsigned long)(strlen(pPartHeaderRLC2) + 1),
                               pPartHeaderRLC2);

    /* Insert parm block B pointer to MAIN parm block */
    gc_util_insert_parm_val(&pParmBlockMain,
                            IPSET_MIME_ACK_TO_REJECTION,
                            IPPARM_MIME_PART,
                            sizeof(unsigned long),
                            (unsigned long)pParmBlockRLC_B);
        frc = gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev,pParmBlockMain,GC_SINGLECALL);
    if(GC_SUCCESS != frc)

    {
        sprintf(str, "gc_SetUserInfo failed for ACK MIME");
        printandlog(index, GC_APIERR, NULL, str, 0);
    }

    gc_util_delete_parm_blk(pParmBlockRLC_B);
    gc_util_delete_parm_blk(pParmBlockMain);
}
```

1.11.5 Exception INVITE/CANCEL/487 Use Case

As described in the Feature Description section, there is a particular use case where the ACK will not contain an embedded MIME body even if the feature was enabled, and the IPSET_MIME_ACK_TO_REJECTION was set ahead of time. Since the call cancellation was already process by the network (ISUP:RLC in 200OK example below), there is no further need to embed a MIME body in the ACK that completes the transaction. The

diagram below shows this specific scenario specifically for a possible SIP <-> ISUP interworking configuration.;



1.12 Increase in Channel Density for G.711 Codec

Service Update 240, increases support for up to 1500 channels of play and record for the G.711 codec. For additional channel density information, select Dialogic® HMP Software 4.1LIN from this location:
http://www.dialogic.com/products/ip_enabled/hmp_software.htm.

1.13 Monitor Mode Support for HMP Conferencing

With Service Update 237, the application is able to form half-duplex audio connections from multiple devices listening half duplex to a single conference party and from multiple conference parties listening half duplex to a single device.

1.13.1 Feature Description

Prior to this feature, there was no way to create multiple half-duplex connections involving a conference party. This limitation prevented customers from making such connections through their “connection of choice” and implementing many use cases. This also limited the number of channels “listening” to a conference to the number of conference resources in a license instead of the number of channels in the system.

1.13.2 Changes to the Conferencing API Library

To implement this feature, three new API functions **cnf_Listen( )**, **cnf_UnListen( )** and **cnf_GetXmitSlot( )**. The data structure, SC_TSINFO, is used for this feature.

Notes:1. The **cnf_Listen( )** and **cnf_UnListen( )** API functions are only recommended for use with “cnf” audio-only conference devices, and have only been validated for use with such devices. For “mcx” multimedia conference devices, **dev_PortConnect( )** and **dev_PortDisconnect( )** are the recommended connection API functions.

2. Although the **dev_Connect( )** and **dev_Disconnect( )** API functions are also supported for the “cnf” audio-only conference devices, it is recommended that the two type of connection methods, **dev_Connect( )/dev_Disconnect( )** and **cnf_Listen( )/cnf_UnListen( )**, should not be used simultaneously. If they are, then the application must take extreme caution to insure that the connections are properly managed.

The descriptions are included in this section.

Name:	int cnf_Listen(a_PtyHandle, a_pTimeslotInfo, a_pUserInfo)	
Inputs:	SRL_DEVICE_HANDLE a_PtyHandle	• valid party handle
	SC_TSINFO *a_pTimeslotInfo	• pointer to TDM bus time slot information structure
	void * a_pUserInfo	• pointer to user-define data
Returns:	CNF_SUCCESS on success CNF_ERROR on error	
Includes:	srllib.h cnflib.h	
Category:	TDM routing	
Mode:	Asynchronous	

Description

The cnf_Listen( ) function connects a party receive channel to a TDM bus time slot, using information stored in the SC_TSINFO data structure. The function sets up a half-duplex connection. For a full-duplex connection, the receive channel of the other device must be connected to the party transmit channel.

The cnf_Listen( ) function returns immediately with success before the operation is completed. After the notification event is received, the party receive channel is connected

to the TDM bus time slot. Although multiple party channels may listen (be connected) to the same TDM bus time slot, the receive channel of a given party device can connect to only one TDM bus time slot.

Parameter	Description
a_PtyHandle	specifies a party device handle obtained from a previous open
a_pTimeslotInfo	specifies a pointer to the SC_TSINFO structure
a_pUserInfo	specifies a pointer to user-defined data

Termination Events

The termination events for this function are:

CNFEV_LISTEN
: Indicates successful completion of this function at which point the party device's receive channel is connected to the TDM bus time slot originally specified in a_pTimeslotInfo.

CNFEV_LISTEN_FAIL
: Indicates indicates the function failed.

Cautions

This function fails when an invalid party handle is specified or when an invalid TDM bus time slot number is specified.

Errors

If the function fails with a CNF_ERROR, use cnf_GetErrorInfo to obtain the reason for the error. Possible errors for this function include:

ECNF_INVALID_DEVICE
: invalid device handle

ECNF_SUBSYSTEM
: internal subsystem error

ECNF_INVALID_PARM
: invalid parameter

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>

// Dialogic headers
#include "srllib.h"
#include "dxxxlib.h"
#include "cnflib.h"

#define MAX_DEVNAME100
#define SRWAITTIMEOUT 10000

long ProcessEvt();

int main(int argc, char* argv[])
```

```
{
    char cnfbdname[MAX_DEVNAME] = {"cnfB1"};
    char dxdevname[MAX_DEVNAME] = {"dxxxB1C1"};
    long devh = -1;
    SRL_DEVICE_HANDLE cnfbdh = -1;
    SRL_DEVICE_HANDLE cnfh = -1;
    SRL_DEVICE_HANDLE cnfptyh = -1;
    long ts;
    SC_TSINFO scts;
    int mode = SR_POLLMODE;

    /* Set SRL to run in polled (non-signal) mode */
    if( sr_setparm( SRL_DEVICE, SR_MODEID, &mode ) == -1 )
    {
        printf( "Error: cannot set srl mode\n" );
        exit( 1 );
    }

    cnfbdh = cnf_OpenEx(cnfbdname, NULL, NULL, EV_SYNC);
    if (cnfbdh == -1)
    {
        printf("Error during call to cnf_OpenEx\n");
        /* perform error processing */
        exit(1);
    }

    /* open conferences */
    cnfh = cnf_OpenConference(cnfbdh, NULL, NULL, NULL);
    if (cnfh == -1)
    {
        printf("Error during call to cnf_OpenConference\n");
        /* perform error processing */
        exit(1);
    }

    if(sr_waitevt(SRWAITTIMEOUT) != -1)
    {
        if (!ProcessEvt())
        {
            /* perform error processing */
            exit(1);
        }
    }
    else
    {
        printf("Error during call to sr_waitevt\n");
        /* perform error processing */
        exit(1);
    }

    cnfptyh = cnf_OpenParty(cnfbdh, NULL, NULL, NULL);
    if (cnfptyh == -1)
    {
        printf("Error during call to cnf_OpenParty\n");
        /* perform error processing */
        exit(1);
    }

    if(sr_waitevt(SRWAITTIMEOUT) != -1)
    {
        if (!ProcessEvt())
        {
            /* perform error processing */
            exit(1);
        }
    }
    else
```

```
{
    printf("Error during call to sr_waitevt\n");
    /* perform error processing */
    exit(1);
}

/* open a voice device */
devh = dx_open(dxdevname, 0);
if (devh == -1)
{
    printf("Error during call to dx_open\n");
    /* perform error processing */
    exit(1);
}

scts.sc_numts = 1;
scts.sc_tsarrayp = &ts;

if (dx_getxmitslot(devh, &scts) == -1)
{
    printf("Error during call to dx_getxmitslot\n");
    /* perform error processing */
    exit(1);
}

printf("Voice device %s (devh=%ld) is transmitting on %ld\n",
       ATDV_NAMEP(devh), devh, ts);

if (cnf_Listen(cnfptyh, &scts, NULL) == -1)
{
    printf("Error during call to cnf_Listen\n");
    /* perform error processing */
    exit(1);
}

printf("Successful call to cnf_Listen\n");

if(sr_waitevt(SRWAITTIMEOUT) != -1)
{
    if (!ProcessEvt())
    {
        /* perform error processing */
        exit(1);
    }
}
else
{
    printf("Error during call to sr_waitevt\n");
    /* perform error processing */
    exit(1);
}

if (cnf_UnListen(cnfptyh, NULL) == -1)
{
    printf("Error during call to cnf_UnListen\n");
    /* perform error processing */
    exit(1);
}

printf("Successful call to cnf_UnListen\n");

if(sr_waitevt(SRWAITTIMEOUT) != -1)
{
    if (!ProcessEvt())
    {
        /* perform error processing */
        exit(1);
```

```
        }
    }
    else
    {
        printf("Error during call to sr_waitevt\n");
        /* perform error processing */
        exit(1);
    }

    return 0;
}

long ProcessEvt()
{
    long ret = 1;
    int devh;
    int evttype;
    long evtlen;
    void* datap;

    printf("ProcessEvt()\n");

    devh = sr_getevtdev();
    evttype = sr_getevttype();
    evtlen = sr_getevtlen();
    datap = sr_getevtdatap();

    switch(evttype)
    {
    case CNFEV_OPEN_CONF:
        printf("Received CNFEV_OPEN_CONF\n");
        break;

    case CNFEV_OPEN_CONF_FAIL:
        printf("Received CNFEV_OPEN_CONF_FAIL\n");
        ret = 0;
        break;

    case CNFEV_OPEN_PARTY:
        printf("Received CNFEV_OPEN_PARTY\n");
        break;

    case CNFEV_OPEN_PARTY_FAIL:
        printf("Received CNFEV_OPEN_PARTY_FAIL\n");
        ret = 0;
        break;

    case CNFEV_LISTEN:
        printf("Received CNFEV_LISTEN\n");
        break;

    case CNFEV_LISTEN_FAIL:
        printf("Received CNFEV_LISTEN_FAIL\n");
        ret = 0;
        break;

    case CNFEV_UNLISTEN:
        printf("Received CNFEV_UNLISTEN\n");
        break;

    case CNFEV_UNLISTEN_FAIL:
        printf("Received CNFEV_UNLISTEN_FAIL\n");
        ret = 0;
        break;

    default:
        printf("Unhandled event: devh(%d); evttype(0x%x)", devh, evttype);
```

```
            break;
        }

        return ret;
    }
```

See Also

- cnf_GetXmitSlot( )
- cnf_UnListen( )

Name:	int cnf_UnListen(a_PtyHandle, a_pUserInfo)	
Inputs:	SRL_DEVICE_HANDLE a_PtyHandle	• valid party handle
	void * a_pUserInfo	• pointer to user-define data
Returns:	CNF_SUCCESS on success CNF_ERROR on error	
Includes:	srllib.h cnflib.h	
Category:	TDM routing	
Mode:	Asynchronous	

Description

The cnf_UnListen( ) function disconnects the conference party receive channel from the TDM bus.

Calling the cnf_Listen( ) function to connect to a different TDM bus time slot automatically breaks an existing connection. Thus, when changing connections, there is no need to call the cnf_UnListen( ) function first.

Parameter	Description
a_PtyHandle	specifies a party device handle obtained from a previous open
a_pUserInfo	specifies a pointer to user-defined data

Termination Events

The termination events for this function are:

CNFEV_UNLISTEN
: Indicates successful completion of this function at which point the party device's receive channel is connected to the TDM bus time slot originally specified in a_pTimeslotInfo.

CNFEV_LISTEN_FAIL
: Indicates indicates the function failed.

Cautions

This function fails when an invalid party handle is specified.

Errors

If the function fails with a CNF_ERROR, use cnf_GetErrorInfo to obtain the reason for the error. Possible errors for this function include:

ECNF_INVALID_DEVICE
invalid device handle

ECNF_SUBSYSTEM
internal subsystem error

Example

For an example of this function, see the example for cnf_Listen ( ).

See Also

- cnf_GetXmitSlot( )
- cnf_Listen( )

Name:	int cnf_GetXmitSlot(a_PtyHandle, a_pTimeslotInfo)	
Inputs:	SRL_DEVICE_HANDLE a_PtyHandle	• valid party handle
	SC_TSINFO *a_pTimeslotInfo	• pointer to TDM bus time slot information structure
Returns:	CNF_SUCCESS on success CNF_ERROR on error	
Includes:	srllib.h cnflib.h	
Category:	TDM routing	
Mode:	Synchronous	

Description

The cnf_GetXmitSlot( ) function returns the time division multiplexing (TDM) bus time slot number of the conference party transmit channel. The TDM bus time slot information is contained in an SC_TSINFO structure that includes the number of the TDM bus time slot connected to the conference party transmit channel.

Parameter	Description
a_PtyHandle	specifies a party device handle obtained from a previous open
a_pTimeslotInfo	specifies a pointer to the SC_TSINFO structure

Cautions

This function fails when an invalid party handle is specified or when an invalid TDM bus time slot number is specified.

Errors

If the function fails with a CNF_ERROR, use cnf_GetErrorInfo to obtain the reason for the error. Possible errors for this function include:

ECNF_INVALID_DEVICE
invalid device handle

ECNF_SUBSYSTEM
internal subsystem error

ECNF_INVALID_PARM
invalid parameter

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>

// Dialogic headers
#include "srllib.h"
#include "cnflib.h"

#define MAX_DEVNAME100
#define SRWAITTIMEOUT 10000

long ProcessEvt();

int main(int argc, char* argv[])
{
    char cnfbdname[MAX_DEVNAME] = {"cnfB1"};
    SRL_DEVICE_HANDLE cnfbdh = -1;
    SRL_DEVICE_HANDLE cnfh = -1;
    SRL_DEVICE_HANDLE cnfptyh = -1;
    long ts;
    SC_TSINFO scts;
    int mode = SR_POLLMODE;

    /* Set SRL to run in polled (non-signal) mode */
    if( sr_setparm( SRL_DEVICE, SR_MODEID, &mode ) == -1 )
    {
        printf( "Error: cannot set srl mode\n" );
        exit( 1 );
    }

    cnfbdh = cnf_OpenEx(cnfbdname, NULL, NULL, EV_SYNC);
    if (cnfbdh == -1)
    {
        printf("Error during call to cnf_OpenEx\n");
        /* perform error processing */
        exit(1);
    }

  /* open conferences */
    cnfh = cnf_OpenConference(cnfbdh, NULL, NULL, NULL);
    if (cnfh == -1)
    {
        printf("Error during call to cnf_OpenConference\n");
        /* perform error processing */
        exit(1);
    }

    if(sr_waitevt(SRWAITTIMEOUT) != -1)
```

```
    {
        if (!ProcessEvt())
        {
            /* perform error processing */
            exit(1);
        }
    }
    else
    {
        printf("Error during call to sr_waitevt\n");
        /* perform error processing */
        exit(1);
    }

    cnfptyh = cnf_OpenParty(cnfbdh, NULL, NULL, NULL);
    if (cnfptyh == -1)
    {
        printf("Error during call to cnf_OpenParty\n");
        /* perform error processing */
        exit(1);
    }

    if(sr_waitevt(SRWAITTIMEOUT) != -1)
    {
        if (!ProcessEvt())
        {
            /* perform error processing */
            exit(1);
        }
    }
    else
    {
        printf("Error during call to sr_waitevt\n");
        /* perform error processing */
        exit(1);
    }

    scts.sc_numts = 1;
    scts.sc_tsarrayp = &ts;

    if (cnf_GetXmitSlot(cnfptyh, &scts))
    {
        printf("Error during call to cnf_GetXmitSlot\n");
        /* perform error processing */
        exit(1);
    }

    printf("Party %s (cnfptyh=%ld) is transmitting on %ld\n", ATDV_NAMEP(cnfptyh), cnfptyh, ts);
    return 0;
}

long ProcessEvt()
{
    long ret = 1;
    int devh;
    int evttype;
    long evtlen;
    void* datap;

    printf("ProcessEvt()\n");

    devh = sr_getevtdev();
    evttype = sr_getevttype();
    evtlen = sr_getevtlen();
    datap = sr_getevtdatap();

    switch(evttype)
```

```
    {
    case CNFEV_OPEN_CONF:
        printf("Received CNFEV_OPEN_CONF\n");
        break;

    case CNFEV_OPEN_CONF_FAIL:
        printf("Received CNFEV_OPEN_CONF_FAIL\n");
        ret = 0;
        break;

    case CNFEV_OPEN_PARTY:
        printf("Received CNFEV_OPEN_PARTY\n");
        break;

    case CNFEV_OPEN_PARTY_FAIL:
        printf("Received CNFEV_OPEN_PARTY_FAIL\n");
        ret = 0;
        break;

    default:
        printf("Unhandled event: devh(%d); evttype(0x%x)", devh, evttype);
        break;
    }

    return ret;
}
```

See Also

- cnf_Listen( )
- cnf_UnListen( )

SC_TSINFO Data Structure

```
typedef struct sc_tsinfo {
        unsigned long sc_numts;
        long *sc_tsarrayp;
} SC_TSINFO;
```

Description

This structure defines the TDM bus (CT Bus) time slot information. It is used by the cnf_GetXmitSlot( ) and cnf_Listen( ) functions.

Field Descriptions

The fields of the CNF_PARTY_INFO data structure are described as follows:

sc_numts
: specifies the number of time slots to follow; must be set to 1 for this release.

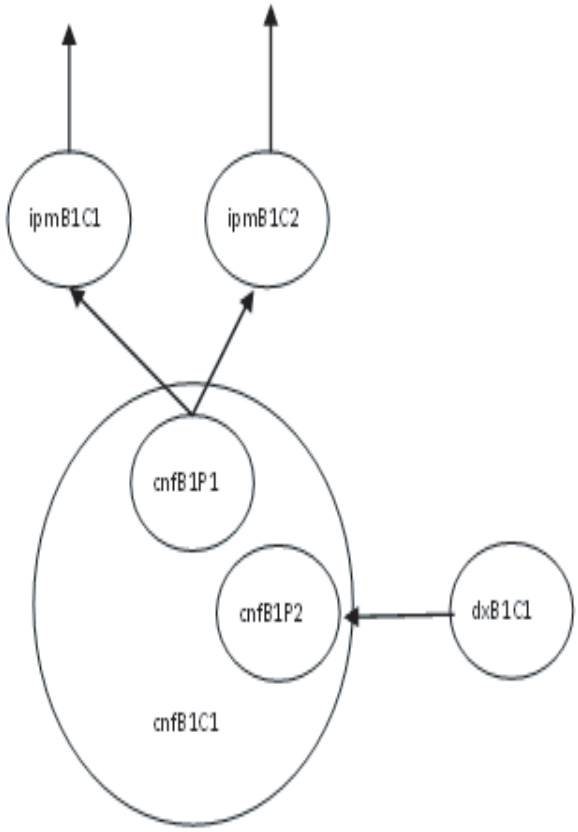
sc_tsarrayp
: specifies the time slot ID number.

1.13.3 Use Cases

This section provides two possible use cases for reference.

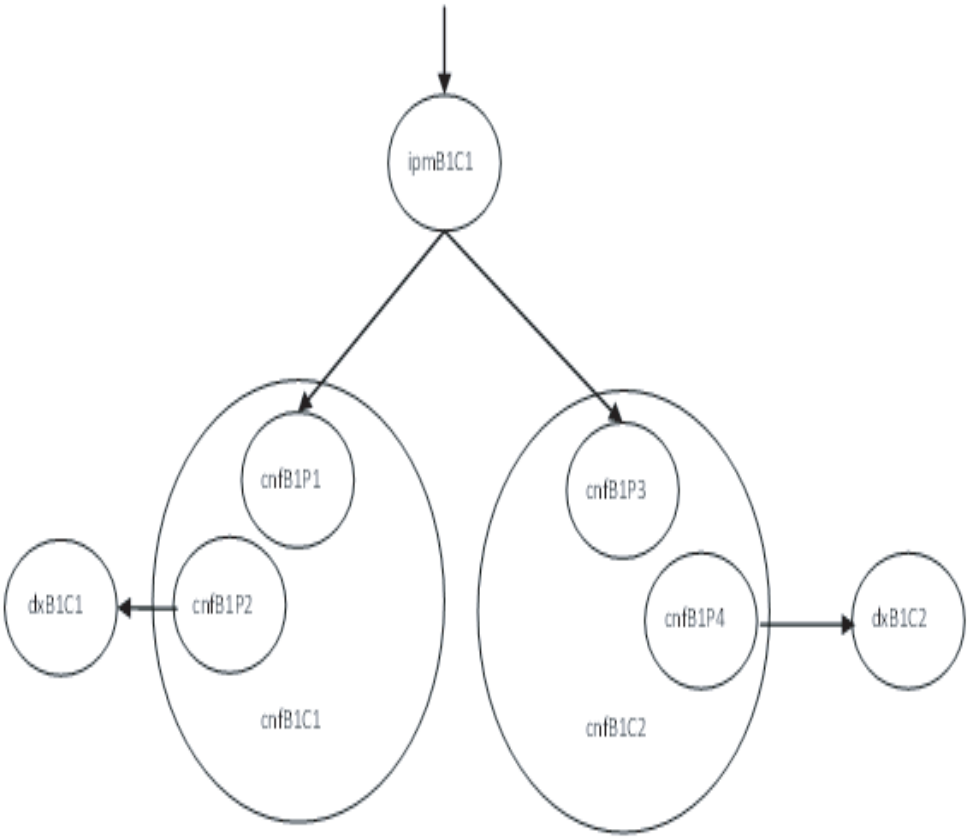
Multiple IPM devices listen to a single conference party

In this use case, multiple externally facing devices, such as IPM devices, make half duplex listening connections to a conference party. Here, the summed media from the conference can be broadcast to the external connection by using only one conference party. In the illustration below, the arrows designate the direction of the data.



Multiple conference parties listen to a single IPM device

In this use case, multiple conference parties in different conferences make half-duplex connections to a single device such as an IPM device. Here, the single inbound media stream from the external connection can be summed into two different conferences.



1.14 Retrieving SIP Inbound RFC 2833

With Service Update 237, the application has the ability to retrieve the RFC 2833 payload type value specified by a remote SIP user agent, using Global Call first party call control (1PCC). Since the ability to set the RFC 2833 payload type on outgoing media streams is already available in 1PCC, applications can take advantage of this new feature to match the outgoing RFC 2833 payload type with the RFC 2833 payload type of the incoming media stream, if its mapping is available in an incoming Session Description Protocol (SDP).

1.14.1 Feature Description

In a typical RFC 2833 application, the IP gateway detects DTMF on the incoming circuits and converts them as RTP payload as described in the RFC 2833, instead of embedding the digits as part of a regular RTP audio payload. On the opposite direction, the gateway recreates the DTMF tones injecting them into PSTN circuits.

RFC 2833 specifies the use of a dynamic payload type and as such its mapping is specified in advance of the media session. The RFC 2833 payload type is described in a SDP media attribute in a SIP message. While it is possible to set the RFC 2833 payload type mapping of an outgoing SDP with Global Call 1PCC mode, it was not possible to retrieve it from an incoming SDP.

With this feature, the application can retrieve the incoming RFC 2833 payload type on a call (CRN) basis using the existing API **gc_GetUserInfo( )** with set ID: IPSET_DTMF and parameter ID: IPPARM_DTMF_RFC2833_PAYLOAD_TYPE. If available, the RFC 2833 payload type of the remote User Agent Client (UAC) can be retrieved on the inbound side upon receipt of a GCEV_OFFERED event. Alternatively the outbound side can retrieve the remote User Agent Server (UAS) RFC 2833 payload type upon receipt of a GCEV_CONNECTED event.

To enable the feature, the application follows the same procedure for the manipulation of the local RFC2833 payload type by calling the **gc_SetUserInfo( )** function with set ID: IPSET_DTMF and parameter ID: IPPARM_SUPPORT_DTMF_BITMASK with the value set as IP_DTMF_TYPE_RFC_2833 after an IPT network device is opened; that is, after the GCEV_OPENEX event is received on the channel device (logical board number and logical channel number). Once this is accomplished, the application can set its own RFC 2833 payload type and now also retrieve the remote RFC 2833 payload type.

Note: If an incoming SDP does not contain RFC 2833 payload type mapping, the **gc_GetUserInfo( )** function will return the IPERR_INVALID_STATE error code. This error code is retrieved by calling the **gc_ErrorInfo( )** function.

For more information about Global all APIs, refer to the Dialogic® Global Call IP Technology Guide, Dialogic® Global Call API Programming Guide, and the Dialogic® Global Call API Library Reference.

Code Examples

The following example shows how to enable RFC2833 payload type manipulation:

```
gc_util_insert_parm_val(&parmblkp, IPSET_DTMF, IPPARM_SUPPORT_DTMF_BITMASK,
                        sizeof(char), IP_DTMF_TYPE_RFC_2833);

if (gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev, parmblkp, GC_ALLCALLS) != GC_SUCCESS) {
          // process error
}
    gc_util_delete_parm_blk(parmblkp);
```

The following example shows how to retrieve the remote payload type from an incoming SIP message SDP carrying the appropriate media attribute with the RFC 2833 payload type mapping in an INVITE.

```
int payloadType = 0;
      case GCEV_OFFERED:
           INIT_GC_PARM_DATA_EXT(&parmdata);
           gc_util_insert_parm_ref(&infoparmblkp,
                                        IPSET_DTMF,
                                        IPPARM_DTMF_RFC2833_PAYLOAD_TYPE,
                                        sizeof(int),
                                        &payloadType );
```

```
if(gc_GetUserInfo(GCTGT_GCLIB_CRN, pline->call[pline->index].crn, &infoparmblkp) !=
   GC_SUCCESS) {
      // Process error
}

while (t_gcParmDatap = gc_util_next_parm(infoparmblkp, t_gcParmDatap)){
       if(t_gcParmDatap->set_ID == IPSET_DTMF && t_gcParmDatap->parm_ID ==
          IPPARM_DTMF_RFC2833_PAYLOAD_TYPE){
          memcpy(&payloadType, (char*)t_gcParmDatap->value_buf, t_gcParmDatap-
                 >value_size);
         printf("Payload type retrieved for the CRN= %ld payload = %d\n ", port
                 [index].crn, payloadType);
       }

       if(payloadType < 96 && payloadType > 127){
           // Invalid payload type range; process error
      }
}
gc_util_delete_parm_blk(infoparmblkp);
 break;
```

Limitations

The following limitations apply:

- To retrieve the RFC 2833 payload type, the **gc_GetUserInfo( )** function can only be called on CRN. If passed an IPT board device or network device handle, the function will return an error.
- An application can call the **gc_GetUserInfo( )** function any time after receiving a GCEV_OFFERED event (inbound) or GCEV_CONNECTED (outbound) event (outbound). Calling the function prior to receipt of the events will return the error IPERR_INVALID_STATE.

1.15 3PCC Support for Dynamic Selection of Outbound SIP Proxy

Service Update 237 adds third-party call control (3PCC) mode for the dynamic selection of outbound SIP proxy feature. Previously, this feature was available in 1PCC mode only. Once enabled in 3PCC, this feature applies to outgoing SIP requests ACK, INFO, INVITE, OPTIONS, REFER, REGISTER, BYE, NOTIFY, SUBSCRIBE, UPDATE and PRACK. For details about this feature, refer to Section 1.16, “Support for Dynamic Selection of Outbound SIP Proxy”, on page 58.

1.16 Support for Dynamic Selection of Outbound SIP Proxy

With Service Update 237, the application can select an outbound SIP proxy server on the Dialogic® HMP virtual board device dynamically. If an outbound SIP proxy server was selected at board initialization it will be overridden, otherwise it will be selected for the first time.

1.16.1 Feature Description

Currently, the outbound proxy address setting is established statically as the Dialogic® HMP board is initialized by setting a specific outbound proxy IP address or a host name in the IP_VIRTBOARD structure that is used by the **gc_Start( )** function. Afterwards, all outgoing SIP messages will pass through the configured outbound proxy. While this method remains in effect, the ability to temporarily and dynamically select the outbound proxy IP address is introduced with this feature.

The application can now configure (or reconfigure) the proxy address to use temporarily for routing all future SIP messages on a virtual HMP board basis. The temporary proxy can be deselected dynamically to revert the outbound proxy setting to the prior setting. If a prior setting was set at virtual board initialization time, then it takes effect immediately; otherwise outbound proxy is disabled.

When the set ID and parameter ID pairs are passed to the Dialogic® HMP virtual board via the **gc_SetConfigData( )** function, outbound messages on all IP channels are routed via the newly set outbound proxy server. The same method is used to deselect the alternate outbound proxy. Once the outbound proxy selection is in place, all supported SIP requests and all outbound SIP responses, within a dialog or not, are affected.

Note: A Contact header field in a 200OK response from the called party (User Agent Server, or UAS) resulting from a Dialogic® HMP INVITE will cause the ACK message and all future requests out of Dialogic® HMP to bypass the proxy and go directly to the UAS for the remaining of the dialog.

Once enabled, this feature applies to the following outgoing SIP requests, in addition to SIP responses out of Dialogic® HMP Software.

- 1PCC mode
  ACK, INFO, INVITE, OPTIONS, REFER, REGISTER, BYE, NOTIFY, SUBSCRIBE and CANCEL.
- 3PCC mode
  ACK, INFO, INVITE, OPTIONS, REFER, REGISTER, BYE, NOTIFY, SUBSCRIBE, UPDATE, PRACK and CANCEL.

Limitations

The following limitations apply to this feature:

- The explicit manipulation of Route and Record-Route headers in the application will override the proxy settings.
- This feature is disabled by default. In order to have the ability of setting an alternate proxy dynamically, explicit feature enablement must be done at virtual board initialization time, using the IP_VIRTBOARD structure in the **gc_Start( )** function.
- The new version of the IP_VIRTBOARD data structure must used to enable this feature.

1.16.2 API Changes to Enable Proxy Override

To allow for backwards compatibility, a new version of the IP_VIRTBOARD data structure is introduced as indicated below. Setting the IP_VIRTBOARD structure to this version makes the new mask, dynamic_outbound_proxy_mask, available to the application.

In addition, a new IP_PROXY_INFO data structure is added to the API. This data structure is used with the **gc_SetConfigData( )** function to dynamically select or deselect the alternate outbound proxy. The in-line function call INIT_IP_PROXY_INFO( ) is used to initialize this data structure.

New Version Define

```
#define VIRTBOARD_VERSION_DYNAMIC_OUTBOUND_PROXY_SUPPORT   0x10f
```

New Mask

```
typedef struct
{
        unsigned short     version;
        unsigned int       total_max_calls;
        unsigned int       h323_max_calls;
        unsigned int       sip_max_calls;
        IP_ADDR localIP;
        ……
        ……
        EnumSIP_Enabled E_SIP_UPDATE_Access;
        unsigned int      dynamic_outbound_proxy_mask;
} IP_VIRTBOARD;
```

Once the feature is enabled, the dynamic outbound proxy can be selected or deselected at any time using an existing set ID and two new parameter IDs as detailed in the following sections.

New Data Structure

```
typedef struct
{
        unsigned long           ulVersion;
        unsigned short          usPort;
        EnumProxyProtocol       eProtocol;
        EnumProxyAddressType    eAddrType;
        char                    arrAddr[CCLIB_SIP_TRANSPORT_IPSTRING_LEN];
        char                    arrHostName[CCLIB_SIP_MAX_HOST_NAME_LEN];
} IP_PROXY_INFO, *pIP_PROXY_INFO;
```

Field Descriptions

The fields of IP_PROXY_INFO are described as follows:

ulVersion
identifies the version of the data structure implementation

usPort
identifies the port to which the proxy protocol is assigned

eprotocol
identifies the proxy protocol

eAddrType
qualifies the eProtocol field; use ePROXY_ADDRESS_TYPE_IP

arrAddr
used to enter a hard-coded proxy IP address

arrHostName
sets the specified hostname as the SIP outbound proxy instead of arrAddr. If arrAddr is set to 0, this hostname is resolved as the outbound proxy address; otherwise this field is ignored. The default value is NULL.

New Parameter Defines

The capability to override the proxy is accomplished by the set ID IPSET_PROXY_INFO, used for configuring proxy information, and two new parameter IDs IPPARM_PROXY_ACTION and IPPARM_PROXY_INFO. The set ID and parameter ID pair are inserted into the GC_PARM_BLKP data structure. Two new values, IP_PROXY_SELECT and IP_PROXY_DESELECT, are added to IPPARM_PROXY_ACTION. The IP_PROXY_SELECT value instructs the library to dynamically choose the proxy, and the IP_PROXY_DESELECT value deselects it. The following table shows the parameter IDs in the IPSET_PROXY_INFO parameter set.

Parameter ID	Description	SIP/H.323
IPPARM_PROXY_ACTION (0x01)	Determines whether to use the proxy for the specific request. Valid values: IP_PROXY_USE 0x00 IP_PROXY_BYPASS 0x01 IP_PROXY_SELECT 0x02 IP_PROXY_DESELECT 0x03	SIP only
IPPARM_PROXY_INFO (BASE_SETID+41)	Denotes the proxy information structure. Valid value: IP_PROXY_INFO data structure	SIP only

1.16.3 Dynamic Proxy Selection Sample Code

This example demonstrates how to enable the feature using the IP_VIRTBOARD structure. Here, the new mask is set to ENUM_Enabled and the version is set to VIRTBOARD_VERSION_DYNAMIC_ OUTBOUND_PROXY_SUPPORT.

```
GC_START_STRUCT gclib_start;
IPCCLIB_START_DATA cclibStartData;
IP_VIRTBOARD virtBoards[1]; // only 1 NIC supported in current release
memset(&cclibStartData,0,sizeof(IPCCLIB_START_DATA));
memset(virtBoards,0,sizeof(IP_VIRTBOARD));

INIT_IP_VIRTBOARD(virtBoards);
virtBoards[0].total_max_calls = g_h323count + g_sipcount;
virtBoards[0].h323_max_calls = g_h323count;
virtBoards[0].sip_max_calls = g_sipcount;
virtBoards[0].sip_signaling_port = g_sipport;
virtBoards[0].sip_msginfo_mask = IP_SIP_MSGINFO_ENABLE | IP_SIP_MIME_ENABLE;
virtBoards[0].E_SIP_OPTIONS_Access = ENUM_Enabled;
virtBoards[0].dynamic_outbound_proxy_mask = ENUM_Enabled;
```

```
virtBoards[0].E_SIP_DefaultTransport = ENUM_UDP;
virtBoards[0].sip_signaling_port =5060;
virtBoards[0].version = VIRTBOARD_VERSION_DYNAMIC_OUTBOUND_PROXY_SUPPORT;

INIT_IPCCLIB_START_DATA(&cclibStartData, 1, virtBoards);
cclibStartData.max_parm_data_size = 512;

CCLIB_START_STRUCT cclib_start[]={{"GC_H3R_LIB", &cclibStartData},
                                  {"GC_IPM_LIB", NULL},
                                  {"GC_DM3CC_LIB", NULL}};

gclib_start.num_cclibs = 2;
gclib_start.cclib_list = cclib_start;

if (gc_Start(&gclib_start) != GC_SUCCESS)
{
     // Handle error
}
```

The following example demonstrates how to set the dynamic proxy information to be selected:

```
// Declare a new structure
IP_PROXY_INFO proxyinfo;

// Open the board device and get the handle
if(gc_OpenEx( &g_sipbd, ":N_iptB1:P_IP", EV_SYNC, NULL) == -1)
{
    // Handle Error
}

// Select the proxy action to select, which will cause the  proxy information to be used in
   setting the outbound details
// Insert this information into the GC Parameter block
rc = gc_util_insert_parm_val(&l_pParmBlk,
                      IPSET_PROXY_INFO,
                      IPPARM_PROXY_ACTION,
                      sizeof(char),
                      IP_PROXY_SELECT);
if (rc < 0)
{
    // Handle error
}

// Initialize the proxy structure using the macro INIT_IP_PROXY_INFO(&proxyinfo);

// Fill in the appropriate values in the structure
strcpy(&(proxyinfo.arrAddr[0]), "146.152.97.100");
proxyinfo.eAddrType = ePROXY_ADDRESS_TYPE_IP;
proxyinfo.eProtocol = ePROXY_PROTOCOL_UDP;
proxyinfo.usPort = 5060;

// Insert the proxy info structure into the GC Parameter Block
rc = gc_util_insert_parm_ref_ex (&l_pParmBlk,
                      IPSET_PROXY_INFO,
                      IPPARM_PROXY_INFO,
                      sizeof(IP_PROXY_INFO),
                      (void*) &(proxyinfo)
                      );
if (rc < 0)
{
     // Handle Error
}

// Use the parameter block from above and send it down to the IPCCLIB
rc = gc_SetConfigData(GCTGT_CCLIB_NETIF,
```

```
                g_sipbd,
                l_pParmBlk,
                0,
                GCUPDATE_IMMEDIATE,
                &l_requestID,
                EV_ASYNC);
if(rc < 0)
{
     // Handle error
}
```

The following example shows how to disable the currently active proxy selection:

```
// Set the appropriate value for the proxy action
rc = gc_util_insert_parm_val(&l_pParmBlk,
                      IPSET_PROXY_INFO,
                      IPPARM_PROXY_ACTION,
                      sizeof(char),
                      IP_PROXY_DESELECT);
if (rc < 0)
{
     // Handle error
}

// Use the parameter block from above and send it down to the IPCCLIB
rc = gc_SetConfigData(GCTGT_CCLIB_NETIF,
                g_sipbd,
                l_pParmBlk,
                0,
                GCUPDATE_IMMEDIATE,
                &l_requestID,
                EV_ASYNC);
if(rc < 0)
{
     // Handle error
}
```

For more information about Global Call IP, see the following documents:

- *Dialogic® Global Call IP Technology Guide*
- *Dialogic® Global Call API Programming Guide*
- *Dialogic® Global Call API Library Reference*

1.17 Defer the Sending of SIP Messages

With Service Update 233, the application can delay sending the appropriate response to an incoming BYE request (such as 200OK), as well as delay the sending of a BYE request until the **gc_ReleaseCallEx( )** function is issued on the channel.

1.17.1 Feature Description

Call termination is accomplished using the **gc_DropCall( )** and **gc_ReleaseCallEx( )** functions. With the SIP protocol, the **gc_DropCall( )** function handles signaling messages by default, while the **gc_ReleaseCallEx( )** is used to free up the resource. For instance, if call termination was initiated by the remote side (the remote side sent the BYE request),

then the **gc_DropCall( )** function will send out a 200OK response. If the call termination was initiated locally, the **gc_DropCall( )** function will send out the BYE request.

When a call is terminated using the **gc_DropCall( )** function, an incoming call could be received before the application has a chance to call the **gc_ReleaseCallEx( )** function to release resources. In situations where there is a lot of call traffic, the incoming call is rejected because even though the last call was dropped, channels are not yet available since the resource has not been released.

By enabling this feature, the **gc_ReleaseCallEx( )** function is responsible for the sending of SIP messages, in addition to the freeing of resources. This allows the application to defer sending the appropriate response to an incoming BYE or CANCEL request or to the BYE and CANCEL requests themselves, or to DECLINE the incoming INVITE, depending on the call state (see Deferring SIP Signaling).

Notes:1. Once enabled, this feature applies to the entire virtual HMP board.

2. The **gc_ReleaseCallEx(**) function is the preferred equivalent to the deprecated **gc_ReleaseCall( )** function.

1.17.2 Deferring SIP Signaling

When **gc_DropCall( )** is issued, GCcclib sends out a SIP message (BYE-Decline/CANCEL/200OK) depending on the state of the call. Prior to this feature, the behavior was as follows:

- Connected state – when **gc_Dropcall( )** is issued to initiate the call termination, the BYE message will be sent out. The GCEV_DROPCALL event will be generated once the 200OK response is received.
- Disconnected state – when the BYE request is received in the connected state, the GCEV_DISCONNECTED will be sent to the application and the state will transition to the Disconnected state. The **gc_DropCall( )** function will send the 200OK response and the GCEV_DROPCALL event will be generated immediately.
- Offered/Accepted state – when **gc_DropCall( )** is issued in the one of these states, the “603-Decline” message will be sent out to reject the incoming call. The GCEV_DROPCALL event will be generated once the ACK is received.
- Dialing state – when **gc_DropCall( )** is issued after **gc_Makecall( )** is issued to initiate the call termination, the CANCEL message will be sent out to cancel the request to start a dialog.

By enabling this feature as described in Enabling SIP Deferral, all the SIP messages will be deferred to the **gc_ReleaseCallEx( )** function. The new behavior is:

- Connected state – when **gc_DropCall( )** is issued, the media session will be terminated and the GCEV_DROPCALL event will be generated. When the **gc_ReleaseCallEx( )** is issued, the BYE message will be sent out. The GCEV_RELEASECALL event will be generated when the 200OK response is received.

- Disconnected state – after the BYE is received in the connected state, the **gc_ReleaseCallEx( )** will send the 200OK final response. The GCEV_RELEASECALL event will be generated immediately.
- Offered/Accepted state – when the **gc_ReleaseCallEx( )** function is issued in the one of these states, the “603-Decline” message will be sent out to reject the incoming call. The GCEV_RELEASECALL event will be generated when the ACK is received.
- Dialing state – when **gc_ReleaseCallEx( )** is issued after **gc_Makecall( )** is issued to initiate the call termination, the CANCEL message will be sent out to cancel the request to start a dialog.

1.17.3 Enabling SIP Deferral

To defer sending SIP messages until the **gc_ReleaseCallEx( )** function is issued, this feature must be enabled for all channels using the **gc_SetConfigData( )** function.

The existing set ID IPSET_CONFIG with the new parameter ID, IPPARM_SIGNALING_DEFERRED, are used to dynamically enable and disable this feature on a virtual board basis. By default, this feature is disabled. This new parameter ID is inserted into the GC_PARM_BLKP data structure using gc_insert_parm_ref( ). The following table shows the new parameter ID in the IPSET_CONFIG parameter set.

Parameter ID	Data Type & Size	Description	SIP/H.323
IPPARM_SIGNALING_DEFERRED	Type: int Size: sizeof(int)	This parameter is used for enabling or disabling the deferral of SIP signaling messages when **gc_DropCall( )** is issued. Possible values are: • GCPV_ENABLE - The signaling messages (BYE/ Decline/CANCEL/200OK, etc) will not be sent out when the **gc_DropCall( )** is issued. They will be deferred to **gc_ReleaseCallEx( )**. • GCPV_DISABLE – Default. The signaling messages (BYE/Decline/CANCEL/200 OK) will be sent when the **gc_DropCall( )** is issued.	SIP only

Example Code

This example demonstrates how to enable the deferral of sending SIP messages during call termination.

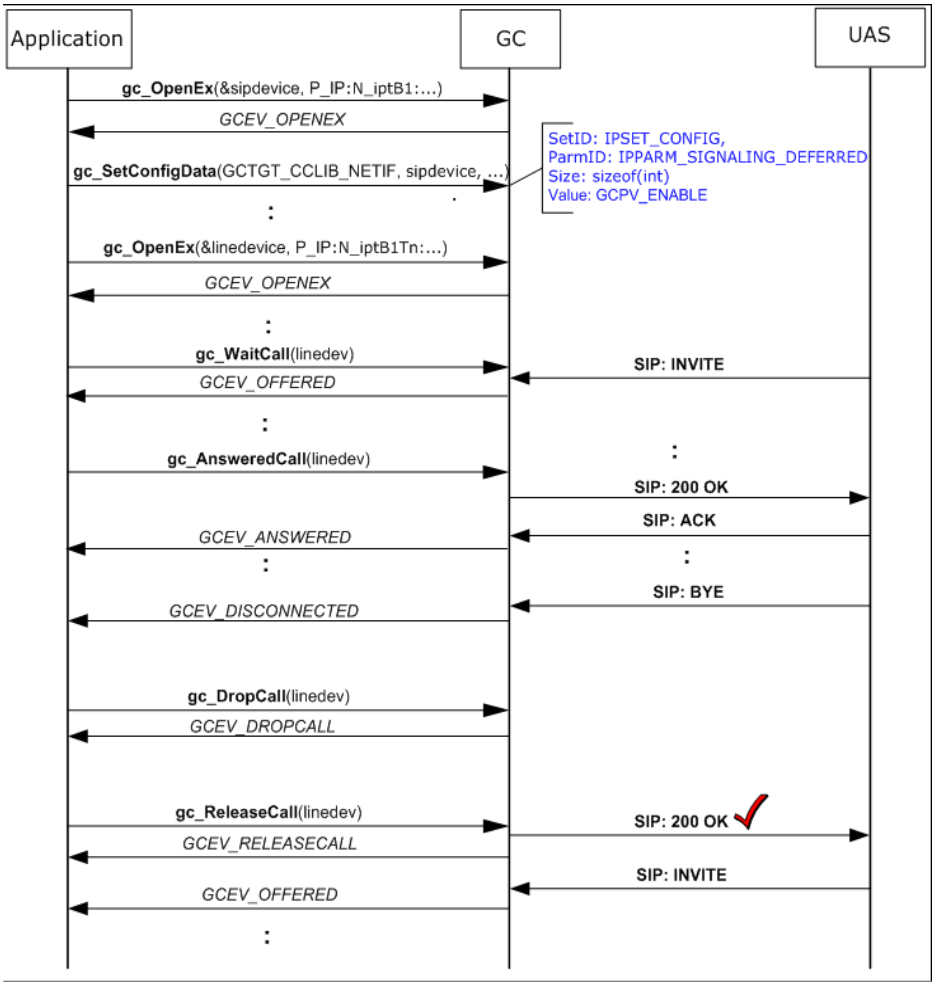
```
#include <stdio.h>
#include <srllib.h>
#include <gclib.h>
#include <gcerr.h>
#include <gcip.h>
#include <gcip_defs.h>

/*
 * Assume the board device iptB1 has been opened
```

```
 */

int defer_signaling()
{
        GC_PARM_BLKP parmblkp = NULL;   /* input parameter block pointer */
        long request_id = 0;
        gc_util_insert_parm_val(&parmblkp,
                IPSET_CONFIG,
                IPPARM_SIGNALING_DEFERRED,
                sizeof(int),
                GCPV_ENABLE);

        if (gc_SetConfigData(GCTGT_CCLIB_NETIF, boardDev, parmblkp, 0 /*timeout*/,
            GCUPDATE_IMMEDIATE, &request_id, EV_ASYNC) != GC_SUCCESS)
        {
            // handle error…
        }

        return (0);
}
```

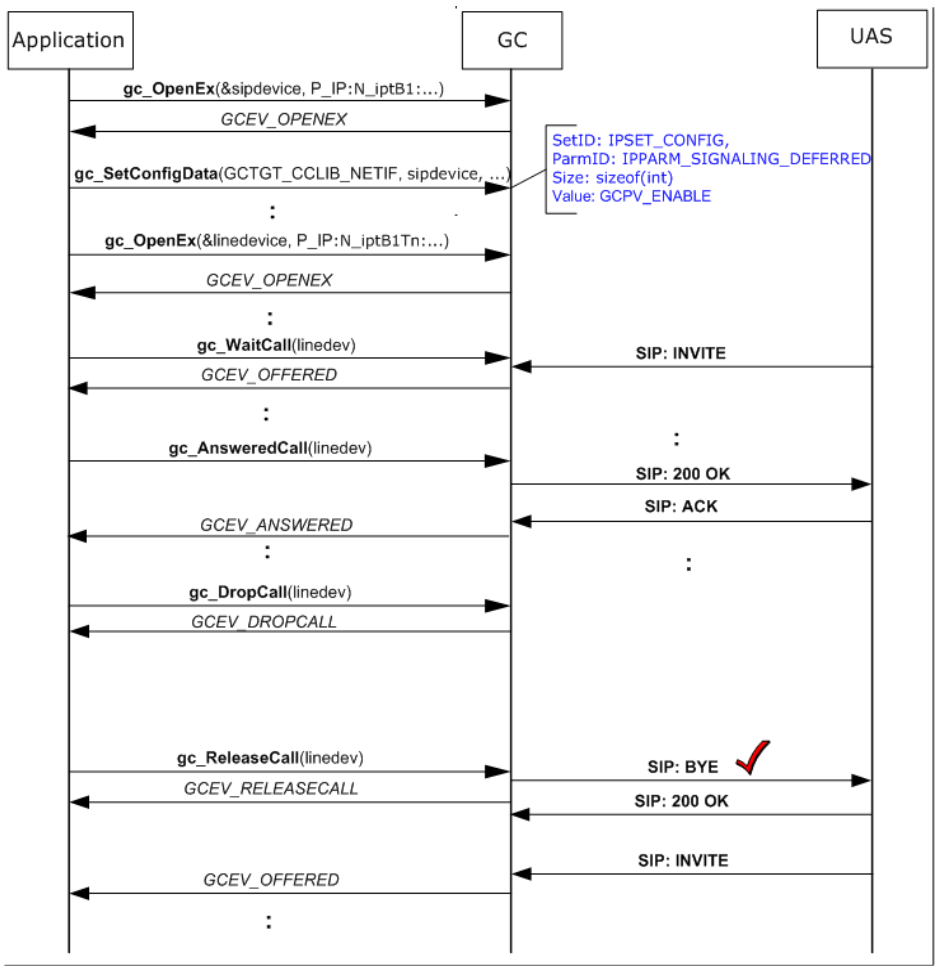
High-level Scenarios

The following diagrams illustrate deferrals of a 200OK response and a BYE request, respectively. Other possible scenarios under different call states are not shown. For

additional information about Global Call IP and basic call control scenarios, see the *Dialogic® Global Call IP Technology Guide.*



The following scenario demonstrates a BYE request.



1.18 Support for WaitCall Cancellation

With Service Update 233, a SIP application is able to dynamically block the ability of an IPT network device to receive a call. This action prevents possible glare conditions during an attempt to make an outbound call while an incoming call is being processed on the same channel. With this feature, the application can block the channel from accepting calls before making an outbound call. If an incoming call is already in progress, the application is notified and the call in progress is not affected.

1.18.1 Feature Description

The Global Call and SIP call control libraries provide IPT network devices (channels) with the ability to receive incoming calls using the **gc_WaitCall( )** function. With IP call control, incoming calls are presented to the application on channels that issued **gc_WaitCall( )** based on an internal algorithm. Prior to this feature, **gc_MakeCall( )** would fail on a channel that was processing an incoming call. To avoid this condition, the application had to call the **gc_Close( )** or **gc_ResetLineDev( )** functions to disable a previously issued **gc_WaitCall( )** function on a channel-by-channel basis. As a result, any incoming call that existed on the line device would be also terminated without notification.

This feature introduces a new API specific to SIP call control. This new **gc_CancelWaitCall( )** function lets the application prevent incoming SIP calls on a particular IPT network device without losing any incoming calls that may have just arrived. The application can use this functionality to cancel any previously issued **gc_WaitCall( )** on a channel-by-channel basis. If the application intends to make an outbound call, it issues **gc_CancelWaitCall( )** prior to the **gc_MakeCall( )** function. The **gc_CancelWaitCall( )** failure indicates that a call is already in progress so it will not be cancelled. When the application receives the error, it does not attempt an outbound call, thus avoiding a possible glare condition.

1.18.2 API Library Changes

The **gc_CancelWaitCall( )** function is introduced to cancel a **gc_WaitCall( )** previously issued on an IPT network device. In addition, the GCEV_CANCELWAITCALL event is added to the library to indicate that notification of incoming calls was successfully disabled.

Name:	int gc_CancelWaitCall(linedev, mode)	
Inputs:	LINEDEV linedev	Global Call line device handle
	unsigned long mode	asynchronous
Returns:	0 if successful <0 if unsuccessful	
Includes:	gclib.h gcerr.h	
Category:	Basic	
Mode:	Asynchronous	
Dialogic® Platform and Technology:	IP† †See the Dialogic® Global Call IP Technology Guides for additional information.	

■ Description

The **gc_CancelWaitCall( )** function cancels any previously issued **gc_WaitCall( )** and disables the ability to receive incoming calls on the given line device.

Notes:1. This function has no effect if the application did not call the **gc_WaitCall( )** function to enable notification of incoming calls. Notification is disabled by default when the channel is opened.

2. The **gc_CancelWaitCall( )** function is currently only supported in asynchronous mode, and only for SIP call control.

Parameter	Description
linedev	Global Call line device handle.
mode	Set to EV_ASYNC for asynchronous execution.

Termination Events

GCEV_CANCELWAITCALL
Indicates that the notification of incoming calls was successfully disabled.

GCEV_TASKFAIL
Indicates that the function failed. For more information, see the "Error Handling" section in the *Dialogic® Global Call API Programming Guide*.

Errors

- If this function returns <0 to indicate failure, use the **gc_ErrorInfo( )** function to retrieve the reason for the error; however, if a GCEV_TASKFAIL event is generated, use the **gc_ResultInfo( )** function instead. See the "Error Handling" section in the *Dialogic® Global Call API Programming Guide*. All Global Call error codes are defined in the *gcerr.h* file, while IP-specific errors codes are specified in *gcip_defs.h*.
- If the line device is processing an incoming call, the function will either fail or generate a GCEV_TASKFAIL event. When the function fails, EGC_GLARE/ IPERR_ADDRESS_IN_USE are set as the GCcclib error codes. If a GCEV_TASKFAIL event is generated, GCEV_CCLIBSPECIFIC /IPEC_InternalReasonIncomingCall are set as the GCcclib cause value.

Example

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

#define MAXCHAN 30   /* max. number of channels in system */
/*
 * Data structure which stores all information for each line
 */
struct linebag {
       LINEDEV ldev; /* line device handle */
       CRN crn; /* GlobalCall API call handle */
       int state; /* state of first layer state machine */
} port[MAXCHAN+1];

struct linebag *pline; /* pointer to access line device */

/*
 * Assume the following has been done:
 * 1. Open line devices for each time slot on iptB1.
 * 2. Each Line Device ID is stored in linebag structure, 'port'.
```

```
 * 3. The gc_WaitCall() has been issued on each Line Device in async mode
 * 4. No call exists on the Line Device
 */

int cancel_waitcall(int port_num)
{
       GC_INFO gc_error_info;
       /* GlobalCall error information data */
       /* Find info for this time slot, specified by 'port_num' */
       pline = port + port_num;
       /*
        * Cancel the wait call
        */
        if (gc_CancelWaitCall(pline->ldev, EV_ASYNC) != GC_SUCCESS) {
              /* process error return as shown */
              gc_ErrorInfo( &gc_error_info );
              printf ("Error: cancel_waitcall() - gc_CancelWaitCall() on device handle: 0x%lx,
                      GC ErrorValue: 0x%hx - %s,CCLibID: %i - %s, CC ErrorValue: 0x%lx -
                      %s\n", pline->ldev, 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);
       }

       return (0);
}
```

For more information about Global Call APIs, see *Dialogic® Global Call API Library Reference* and the *Dialogic® Global Call API Programming Guide*. For IP-specific call control, see the *Dialogic® Global Call IP Technology Guide*.

1.19 Support for Enhanced RTCP Reports

With Service Update 221, the Dialogic® IP Media library supports Quality of Service (QoS) alarms and enhanced RTCP reports (RTCP-XR) compliant with RFC 3611 for RTCP Extended Reports and IETF draft RTCP High Resolution VoIP Metrics Report Blocks (RTCP-HR). For more information about this feature, refer to the *Dialogic® IP Media Library API Programming Guide and Library Reference* available on the documentation bookshelf.

1.20 Support for Streaming H.263 Baseline Profile Using RFC 2429 Packetization

With Service Update 221, the application can natively stream audio and video from an RTSP Server to 3G-324M calls terminated by the Dialogic® product. These 3G-324M calls may be established over the PSTN (for example, E1 circuit-switched connection) or over IP. For more information about this feature, refer to the *Dialogic® IP Media Library API Programming Guide and Library Reference* available on the documentation bookshelf.

1.21 AMR-NB and G.711 Audio Over Nb UP

Service Update 221 adds support for streaming AMR-NB and G.711 audio over Nb UP. Previously only streaming 3G-324M over Nb UP was supported. For more information about this feature, refer to the *Dialogic® IP Media Library API Programming Guide and Library Reference* available on the documentation bookshelf.

1.22 Support for Red Hat Enterprise Linux Release 5.0 Update 2

Service Update 221 adds support for Red Hat Enterprise Linux Release 5.0 Update 2.

Note: When Red Hat Enterprise Linux Release 5.0 Update 2 is installed, select the following file: *libstdc++ version 5 libs.*

1.23 Support for Red Hat Enterprise Linux Release 5.0 Advanced Server, Enterprise Server, or Workstation (AS, ES, or WS)

With Service Update 204, Dialogic® HMP Software 3.1LIN supports Red Hat Enterprise Linux Release 5.0 Advanced Server, Enterprise Server, or Workstation (AS, ES, or WS).

Note: For video play and record, Red Hat 5.0 is supported in NFS and RAID configurations only. A known issue (IPY00044980) exists when running over 60 channels on a system with a local hard drive.

1.24 Enabling Runtime Tracing in 3G-324M API

With Service Update 203, runtime tracing on a per channel or board basis is supported in the 3G-324M API library. For more information about this feature, refer to the *Dialogic® 3G-324M API Programming Guide and Library Reference* available on the documentation bookshelf.

1.25 Multiple Frames Per Packet Support for AMR-NB

With Service Update 201, multiple frames per packet (fpp) are supported for AMR-NB codec for RTP interface: 1 to 10 fpp (octet-aligned and bandwidth-efficient). Previously only 1 fpp was supported.

Note: Using the AMR-NB resource in connection with one or more Dialogic products mentioned herein does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at *http://www.voiceage.com/licensing.php*.

1.26 Support for ATDX_BUFDIGS( ) Function

With Service Update 197, the **ATDX_BUFDIGS( )** function is supported. To enable the **ATDX_BUFDIGS( )** function, add SupportForSignalCounting = 1 in */usr/dialogic/cfg/cheetah.cfg*. To subsequently disable this function, remove this line from the .cfg file. For more detailed information about **ATDX_BUFDIGS( )**, see Section 3.4.21, "Dialogic® Voice API Library Reference", on page 130.

1.27 Retrieving FlexLM-based Licensed Feature Data

With Service Update 192, an application can retrieve information about licensed features existing on the Dialogic® platform before calling the **gc_Start( )** function.

1.27.1 Feature Description

This feature provides the ability to retrieve the exact number of IP Call Control (IPT) devices licensed and available so that the application can pass a number less than or equal to this value to the **gc_Start( )** function. This ability to do this avoids an IPCCLIB startup failure.

The IP Call Control library needs to know the number of H.323 and SIP IPT devices to open on behalf of a IP Call Control user application. Previously, this information passed to the user application via the IPCCLIB_START_DATA structure when calling the **gc_Start( )** function. In first-party Call Control (1PCC) mode, there is an implicit one-to-one correlation between the number of IPT devices (controlled by IPCCLIB) and the number of RTP or IPM devices (controlled by the firmware and licensing scheme). If the application attempted to enable more IPT devices than the number of licensed IPM devices, an error would return indicating a non-associated IPM device.

In 3PCC mode, there is no one-to-one correspondence between IPT and IPM devices. Instead, the application manages the RTP or media streams for associated IP Call Control devices. The RTP devices can be non-Dialogic, so the application is allowed to open more IPT devices than licensed IPM devices. If IP CCLIB checks the number of IPT devices passed into the **gc_Start( )** function against the number of IP Call Control licenses available on the Dialogic® platform, the library fails to start if the application tries to open more IPT devices than the number of licenses available. This feature resolves this limitation.

1.27.2 New Header File, Defines, and Data Structures

This section introduces the changes required to provide the ability to retrieve licensed feature information from the associated Dialogic® Platform, while insulating the data from the FlexLM licensing header files.

A new header file, *LicensedFeatures.h*, is added to the Dialogic® System Release and contains the following defines:

```
#define LM_FEATURE_NAME_BASIC_VOICE    "Voice"
#define LM_FEATURE_NAME_ENHANCED_VOICE "Enhanced_RTP"
#define LM_FEATURE_NAME_CONFERENCE     "Conferencing"
#define LM_FEATURE_NAME_CSP            "Speech_Integration"
#define LM_FEATURE_NAME_FAX            "Fax"
#define LM_FEATURE_NAME_RTP            "RTP_G_711"
#define LM_FEATURE_NAME_MM             "Multimedia"
#define LM_FEATURE_NAME_IPCC           "IP_Call_Control"
#define LM_FEATURE_NAME_AMR            "AMR_NB"
#define LM_FEATURE_NAME_NA             "Native_Audio"
#define LM_FEATURE_NAME_NRTP           "Native_RTP"
#define LM_FEATURE_NAME_MUX3G          "3G-324M"
```

The following defines and data structures are added to the *devmgmt.h* header file:

```
/* FlexLM license info retrieval defines and data structures */

#define MAX_FEATURES      256 /* Max features licensed */
#define LICENSED_FEATURES_VERSION 0x000  /* License Structure version */

typedef struct
{
    LicenseFeatureEnum FeatureEnum;       /* Licensed Feature Enum        */
    unsigned short     FeatureCount;      /* Licensed Feature Total Count */
    unsigned short     rfu;               /* For future use               */
}LIC_FEATURE_DATA;

/*
    Data structure used to return block of flexlm licensed features info.
*/
typedef struct
{
    unsigned long       version;        /* Licensing structure version */
    unsigned int        NumFeatures;    /* Number of features info returned. */
    LIC_FEATURE_DATA    lic_data_buf[MAX_FEATURES];/* License data buffer    */
}LIC_FEATURE_BLK, *PLIC_FEATURE_BLK;

static __inline void INIT_LICENSED_FEATURES_VERSION(PLIC_FEATURE_BLK plic_data)
{
   plic_data->NumFeatures = 0;
   plic_data->version = LICENSED_FEATURES_VERSION;
}
```

1.27.3 Code Example

In this example, the application first initializes the data structure in which the licensing information is returned, and then passes a pointer to the LIC_FEATURE_BLK structure to the **dev_getLicFeatureData( )** function. The returned LIC_FEATURE_BLK data structure contains the number of actual licensed features existing on the Dialogic® platform, as well as each licensed feature's name and count (number of instances).

```
#include "devmgmt.h"

int main()
{
 int rc;
 unsigned long lic_msk;
 LIC_FEATURE_BLK lic_data;
 INIT_LICENSED_FEATURES_VERSION(&lic_data);
 rc = dev_GetLicFeatureData(&lic_data);
 if (rc != 0)
 {
  printf("Could not get licensed feature information...Feature not supported or FlexLM license
```

```
          file is not active\n");
 }
 else
 {
  for (unsigned int i = 0; i < lic_data.NumFeatures; i++)
  {
   printf("\n*******************************");
   printf("\n  Licensed Feature Name: %s",
          FeatureNameTable[lic_data.lic_data_buf[i].FeatureEnum]);
   printf("\n Licensed Feature Count: %d ",lic_data.lic_data_buf[i].FeatureCount);

   if ( 0 == strcmp(FeatureNameTable[lic_data.lic_data_buf[i].FeatureEnum],
      LM_FEATURE_NAME_IPCC))
    printf("\n Number of 3PCC IP Call Control devices licensed = %d
          ",lic_data.lic_data_buf[i].FeatureCount);
  }
 }
}
```

1.28 AMD Machine Support for IP-only Configurations

With Service Update 192, the Dialogic® Host Media Processing Software Release 3.1LIN supports AMD machines for IP-only configurations. The following systems and licenses were tested:

Dual socket dual core Opteron 8220 2.8GHz

- 500r500v200e500c240s0f0i0m_host_pur.lic
- 240r240v120e0c0s0f240i120m_host_pur.lic

1.29 Support for Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards

With Service Update 187, Dialogic® Host Media Processing Software Release 3.1LIN supports the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards. These Dialogic® HMP Interface Boards are high-performance digital network interface boards in a full-length PCI Express form factor. The DNI/310TEPEHMP has one E1/T1 network interface, the DNI/610TEPEHMP has two E1/T1 network interfaces, and the DNI/1210TEPEHMP has four E1/T1 network interfaces.

The use of Dialogic® HMP Interface Boards with Dialogic® HMP Software enables applications developed using Dialogic® API libraries in an HMP environment to connect to PSTN networks through E1 or T1 network interfaces. The Dialogic® HMP Interface Boards can be used for converged IP-PSTN solutions that require both IP and PSTN connectivity.

This section provides information about:

- Features of the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards

- Installing the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards
- Installing the Dialogic® HMP Software
- Obtaining a License File
- Configuring the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards
- Runtime Configuration of Echo Cancellation

1.29.1 Features of the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards

Features of the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards include:

Interface to Dialogic® HMP Software
: Allows host-based voice, speech, conference, fax, and IP transcoding to be accessible from the PSTN interface; can be configured in a wide range of densities, scalable in individual port increments

One, two, or four digital network interfaces
: Provides different densities to support a cost-effective range of solutions

Software-selectable trunks configure DNI Boards for either E1 or T1
: Reduces the total cost of ownership by increasing flexibility, reducing inventory, and simplifying the purchasing process and test effort

Support for a wide range of PSTN protocols including ISDN and CAS signaling
: Allows a choice of PSTN protocols

Dialogic® Global Call Software
: Provides a consistent programming interface for call control utilized by boards with Dialogic® DM3 architecture and by Dialogic® HMP Software

Host streaming interface
: Enables a low-latency, 256-duplex channel interface to host-based media and IP networks

Soft H.100 CT Bus
: Provides a software-defined bus that extends the traditional board-based H.100 CT Bus to host-based media and IP networks

For further information about features, applications, and technical specifications, refer to the product data sheet, which is available at http://www.dialogic.com.

1.29.2 Installing the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards

For board installation instructions, see the Installation Guide (Quick Install Card) that comes with each board. The Installation Guide explains how to set the jumpers on the board, install the board in the computer, and connect to external equipment.

Note: Refer to the Installation Guide for important information about power budgeting and guidelines for selecting the slot where a board can be installed. If the board is not configured and installed properly, it will not be detected in the system.

1.29.3 Installing the Dialogic® HMP Software

The Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards require the use of a Dialogic® HMP Software release that specifically supports these boards. If you are installing one of these boards in a system that already has HMP Software installed, you should verify that your installed software version supports the board. If it does not support the board, you will need to obtain and install a Service Update that does support the board before configuring the system for the newly installed board(s).

Note: When installing the HMP Software, the **DNI Boards & HMP Software** package is required when using Dialogic® HMP Interface Boards. This is one of the items in the menu of packages displayed during the installation process. If the HMP Software has already been installed without this package, uninstall the HMP Software and then reinstall it, selecting the correct package.

To check which packages were installed, enter:

```
rpm -qa lsb-dialogic-hmp-\*
```

The names of the installed packages are listed. The name of the DNI Boards & HMP Software package is **lsb-dialogic-hmp-t1e1**.

For detailed information about installing the HMP Software, see the *Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide*.

1.29.4 Obtaining a License File

Before you use the Dialogic® HMP Software and the supported Dialogic® HMP Interface Boards, you must obtain a license file containing HMP license data. An HMP license is a file containing authorization for a combination of call control and media processing features. You can obtain a license file either before or after you install the HMP Software and supported boards, but you need to obtain a license file before you can proceed with using the HMP Software and supported boards.

If you have been using the HMP Software *without* Dialogic® HMP Interface Boards, you have a **host-based license**, which is associated with the host machine via its host ID

(MAC address). You **cannot** use a host-based license with Dialogic® HMP Interface Boards. Instead, you will need a **board-based license**, which is associated with one of the boards in the system via its board serial number.

If you are adding Dialogic® HMP Interface Boards to a system that already has a board-based license, you can use the existing license but you may have to upgrade it.

For detailed information about obtaining and upgrading a license, see the *Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide*.

1.29.5 Configuring the Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards

After the Dialogic® HMP Software is installed and the appropriate licensing is obtained, you can begin the configuration process. Two tools are available to perform configuration tasks:

- Command Line Interface (CLI)
- Simple Network Management Protocol (SNMP)

Both tools have access to the same configuration data.

The *Dialogic® System Configuration Guide* provides detailed instructions for configuring Dialogic® HMP Software and HMP Interface Boards in the system. When configuring the system for the new PCI Express form factor boards, use the same procedures that are documented for the PCI version of the boards.

If you are using the Dialogic® Global Call protocols, see the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for additional configuration procedures.

Note: In the *pdk.cfg* file, the following mlmfile options are used with the DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Boards:

- **mlmfile hmp_pdk_octal.mlm.sym** if the board is using CAS on all trunks
- **mlmfile hmp_mixed_octal.mlm.sym** if the board is mixing CAS and ISDN protocols

1.29.6 Runtime Configuration of Echo Cancellation

Refer to Enabling Echo Cancellation with Non-Linear Processing for information about runtime configuration of echo cancellation.

1.30 Revised Multimedia User I/O Implementation

Service Update 187 introduces the revised user I/O implementation. User I/O enables applications to directly play and record RTP data via user I/O buffers. For more

information, refer to the revised *Dialogic® Multimedia API Library Reference* available at http://www.dialogic.com/manuals/hmp31lin.

1.31 Additional Operating System Support

With the Service Update 187, Dialogic® HMP Software 3.1LIN supports SUSE Linux Enterprise Server 9 Service Pack 4 and Red Hat Release 4.0 Updates 5 and 6.

1.32 Receive-only RFC2833 Mode Available

This Service Update 187 allows users to specify a receive-only RFC2833 mode. When this mode is specified, the additional audio transmission delay associated with the current full duplex RFC2833 mode, which affects the transmitted audio RTP packets, is eliminated. The additional audio transmission delay exhibited by the current full duplex RFC2833 mode, is the result of a mechanism employed to detect and remove in-band DTMF digits from the audio stream prior to the transmission of audio RTP packets.

1.32.1 API Usage and New IP_PARM Values

For the third-party call control model (3PCC), an application can select this new mode of operation at runtime using the **ipm_SetParm( )** function with the appropriate parameter and value prior to invoking the **ipm_StartMedia( )** function.

For the first-party call control model (1PCC), this new mode will be selected using the same method used to select the current DTMF modes. The current DTMF modes are selected using the parameter element IPSET_DTMF in GC_PARM_BLK, which is associated with the **gc_SetUserInfo( )** and **gc_SetConfigData( )** functions.

The IP_PARM_INFO data structure has been updated to support the receive-only RFC2833 mode. This data structure is used to set or retrieve parameters for an IP channel. The structure is used by the **ipm_GetParm( )** and **ipm_SetParm( )** functions.

eIP_PARM Define	Description and Values
PARMCH_DISABLE_TX_TELEPHONY_EVENT	Disables transmit RFC2833 digits. Values: • 1 = disable transmission • 0 = enable transmission

1.32.2 Code Examples

The following example demonstrates how to set or get parameters using the IP Media API.

```
int handle;
handle = ipm_Open("ipmB1C1,0,EV_ASYNC);
// wait for event IPMEV_OPEN
IPM_PARM_INFO ipmParmInfo;"
int value = 1;    /* disable */
IpmParmInfo.eParm = PARMCH_DISABLE_TELEPHONY_EVENT;
IpmParmInfo.pvParmValue = &value;
ipm_SetParm(handle, &ipmParmInfo, EV_ASYNC);
// wait for event IPMEV_SET_PARM

IpmParmInfo.eParm = PARMCH_DISABLE_TELEPHONY_EVENT;
Ipm_GetParm(handle, &ipmParmInfo, EV_ASYNC);
// wait for event IPMEV_GET_PARM
IPM_PARM_INFO *pParm = (IPM_PARM_INFO *)sr_getevtdatap();
printf("Parameter value is: %d\n", *(pParm->pvParmValue));
```

The following example demonstrates how to set PARMCH_DISABLE_TX_TELEPHONY_EVENT using the Global Call API:

```
int  value  = 1;    // Disable transmit RFC2833 digits
IPM_PARM_INFO ipmParmInfo;
GC_PARM_BLKP parmblkp;

ipmParmInfo.eParm = PARMCH_DISABLE_TX_TELEPHONY_EVENT;
ipmParmInfo.pvParmValue = (void *)&value;

gc_util_insert_parm_ref(&parmblkp, IPSET_CONFIG, IPPARM_IPMPARM, (unsigned
long)sizeof(IPM_PARM_INFO), &ipmParmInfo);
gc_SetUserInfo(GCTGT_GCLIB_CHAN, lineDev, parmblkp, GC_ALLCALLS);
gc_util_delete_parm_blk(parmblkp);
```

For more information, refer to the *Dialogic® Global Call IP Technology Guide* located at http:// www.dialogic.com/manuals.

Only the GC_ALLCALLS mode supports this feature.

1.32.3 Download and Startup Parameter Usage

Add the following line to the IPVSC [0x40] section of the *Hmp.Uconfig* file used to download the Dialogic® HMP board:

```
SetParm=0x4019, 1   ! 1 means Disable transmit RFC2833 digits
```

Stop the Dialogic® Services, and then re-start services to download the board.

1.33 Support for GSM Codec

With Service Update 174, Dialogic® HMP Software 3.1LIN supports the GSM codec. Specifically, GSM full rate at 13 Kbps (ETSI 6.10 and RTP; both Microsoft and TIPHON frame support).

1.34 Support for 3G-324M Release 4 (Nb UP)

With Service Update 157, Dialogic® HMP Software 3.1LIN supports 3G-324M multimedia sessions over IP using the Nb UP protocol as defined in 3GPP Release 4.

1.35 Support for SUSE Linux Enterprise Server 9 Service Pack 3

With Service Update 157, Dialogic® HMP Software 3.1LIN supports SUSE Linux Enterprise Server 9 Service Pack 3 (SLES 9 SP3) with reconfigured Kernel 2.6.5. The following describes additional requirements.

1.35.1 Disabling ACPI Features on SUSE Linux for Dialogic® HMP Interface Boards

If you use Dialogic® HMP Interface Boards in a system with SUSE Linux 9.3, Advanced Configuration and Power Interface (ACPI) features must be disabled. To disable ACPI features, specify the following command in /boot/grub/menu.lst on the kernel line:

```
pci=noacpi
```

Refer to your operating system manual for more information on how to set kernel boot parameters.

For more information on controlling ACPI (Advanced Configuration and Power Interface) and/or APIC (Advanced Programmable Interrupt Controller), see the open SUSE web site at: http://en.opensuse.org/SDB:Kernel_Parameters_for_ACPI/APIC.

1.35.2 Selecting Linux Kernel Source for Dialogic® HMP Interface Boards

If you use Dialogic® HMP Interface Boards in a system with SUSE Linux 9.3 or with Red Hat Linux 4.0 (with Update 1, 2, 3, or 4), source for the Linux kernel (kernel headers) must be installed on your system. This can be done by selecting these packages during the initial OS install or after the initial OS install by installing these source RPMs from your Linux OS distribution media.

On SUSE Linux, after the kernel source is installed, you must reconfigure the kernel source as follows:

- In the /usr/src/linux directory:
  – run `make cloneconfig`
  – run `make modules_prepare`
- Remove symbolic link /lib/modules/'uname -r'/build.
- Create a symbolic link /lib/modules/'uname -r'/build and direct it to /lib/modules/`uname -r'/source.

1.35.3 Reconfiguring Kernel 2.6.5 of SUSE Linux Enterprise Server 9 to work with Dialogic® HMP Software

At a minimum, the kernel .config file should be configured for an Intel Pentium 4 processor (which also includes the Mobile Intel Pentium 4 Processor-M and the Intel Xeon processor) as follows:

- CONFIG_M586 should be disabled and CONFIG_MPENTIUM4 should be enabled
- CONFIG_PREEMPT should be enabled
- CONFIG_HPET_RTC_IRQ should be disabled

After changing the configuration, you must recompile the kernel using the following commands:

- make
- make modules_install
- make install

Although the .config file can be manually configured, it is recommended that you use the make menuconfig command or the provided configuration script from the web site at: http://www.dialogic.com/support/helpweb/hmp/hmplinux/hmp12/suse.

1.36 Support for Dialogic® SS7 Boards

With Service Update 157, Dialogic® HMP Software 3.1LIN supports Dialogic® SS7 boards. Dialogic® SS7 boards provide on-board support for SS7 common channel signaling protocols with a number of digital line interfaces (T1/E1/J1) and a H.100 or H.110 PCM highway that supports connection to a wide range of voice, data, and fax boards. See the Release Guide for supported Dialogic® SS7 boards. Refer to the *Dialogic® Global Call SS7 Technology Guide* for more details on the features and configurations.

1.37 Support for MSML Video, PSTN, and MSML Audit Package

With Service Update 157, Dialogic® HMP Software 3.1LIN supports MSML Video, PSTN, and MSML Audit Package. Refer to the *MSML Media Server Software User's Guide* for additional information.

1.38 Support for Dialogic® DNI2410TEPEHMP Digital Network Interface Board

With the Service Update 154, Dialogic® HMP Software 3.1LIN supports the new Dialogic® DNI2410TEPEHMP Digital Network Interface Board. The DNI2410TEPEHMP Board is a high-density, high-performance, digital network interface board with eight T1/E1 network interfaces in a full-length PCI Express form factor.

The use of digital network interface boards with Dialogic® HMP Software enables applications developed using Dialogic® API libraries in an HMP environment to connect to PSTN networks through T1 or E1 network interfaces. The digital network interface boards can be used for converged IP-PSTN solutions that require both IP and PSTN connectivity.

This section provides information about:

- Installing the Dialogic® DNI2410TEPEHMP Digital Network Interface Board
- Installing the Dialogic® HMP Software
- Licensing
- Configuring the Dialogic® DNI2410TEPEHMP Digital Network Interface Board
- Runtime Configuration of Echo Cancellation

1.38.1 Installing the Dialogic® DNI2410TEPEHMP Digital Network Interface Board

For board installation instructions, see the Installation Guide (Quick Install Card) that comes with each board. The Installation Guide explains how to set the jumpers on the board, install the board in the computer, and connect to external equipment using splitter cables.

Note: Refer to the Installation Guide for important information about power budgeting and guidelines for selecting the slot where a board can be installed. If the board is not configured and installed properly, it will not be detected in the system.

1.38.2 Installing the Dialogic® HMP Software

The Dialogic® DNI2410TEPEHMP Digital Network Interface Board requires the use of a Dialogic® HMP Software release that specifically supports it. If you are installing the DNI2410TEPEHMP Board in a system that already has HMP Software installed, you should verify that your installed software version supports the board. If it does not support the board, you will need to obtain and install a Service Update that does support the DNI2410TEPEHMP Board before configuring the system for the newly installed board(s).

Note: When installing the HMP Software, the **DNI Boards & HMP Software** package is required when using digital network interface boards. This is one of the items in the menu of packages displayed during the installation process. If the HMP Software has already been installed without this package, uninstall the HMP Software and then reinstall it, selecting the correct package.

To check which packages were installed, enter:

```
rpm -qa lsb-dialogic-hmp-\*
```

The names of the installed packages are listed. The name of the DNI Boards & HMP Software package is **lsb-dialogic-hmp-t1e1**.

For detailed information about installing the HMP Software, see the *Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide*.

1.38.3 Licensing

Before you use the Dialogic® HMP Software and the supported digital network interface boards, you must obtain a license file containing HMP license data. An HMP license is a

file containing authorization for a combination of call control and media processing features. You can obtain a license file either before or after you install the HMP Software and supported boards, but you need to obtain a license file before you can proceed with using the HMP Software and supported boards.

If you have been using the HMP Software *without* digital network interface boards, you have a **host-based license**, which is associated with the host machine via its host ID (MAC address). You **cannot** use a host-based license with the Dialogic® DNI2410TEPEHMP Digital Network Interface Board. Instead, you will need a **board-based license**, which is associated with one of the boards in the system via its board serial number.

If you are adding a DNI2410TEPEHMP Board to a system that already has a board-based license, you can use the existing license but you may have to upgrade it.

For detailed information about obtaining and upgrading a license, see the *Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide*.

1.38.4 Configuring the Dialogic® DNI2410TEPEHMP Digital Network Interface Board

After the Dialogic® HMP Software is installed and the appropriate licensing is obtained, you can begin the configuration process. Two tools are available to perform configuration tasks:

- Command Line Interface (CLI)
- Simple Network Management Protocol (SNMP)

Both tools have access to the same configuration data.

The *Dialogic® System Configuration Guide* provides detailed instructions for using these tools to configure HMP Software and digital network interface boards in the system. This section provides supplementary information that is applicable to the Dialogic® DNI2410TEPEHMP Digital Network Interface Board:

- Trunk Configuration Using CLI
- Trunk Configuration Using SNMP
- FCD/PCD File Names

1.38.4.1 Trunk Configuration Using CLI

The “Configuration Procedures Using CLI” chapter in the *Dialogic® System Configuration Guide* includes a procedure for configuring PSTN network interfaces. You can follow that procedure for the DNI2410TEPEHMP Board. The procedure is repeated here, with supplementary information for the DNI2410TEPEHMP Board.

1. Verify that the media services and boards are stopped. See “Stopping Media Services” procedure in the *Dialogic® System Configuration Guide*.

   Note: You can configure PSTN network interface settings while media services are active. The configuration is **updated**; however, the configuration is only **applied** (1) after media services are stopped, (2) the **conf system pstn apply** command is issued (see Step 6), and (3) media services are restarted. Thus, it is recommended that you stop media services before performing the procedure.

2. Retrieve the device ID for the hardware in your system.

   ```
   CLI> show hardware
   ```

   All Dialogic® devices are displayed including device ID, technology, device name, slot ID, and more.

3. Retrieve the current settings for the PSTN network interfaces.

   ```
   CLI> show hardware pstn
   ```

   Information about device ID, trunk ID, interface ID, protocol, and more is displayed.

4. Modify a current PSTN network interface setting for a specific device and trunk; for example, modify protocol.

   Note: This step must be repeated for each trunk; entering a range of trunk IDs is not allowed.

   ```
   CLI> conf hardware pstn <device ID> <trunk ID> protocol <protocol
   value>
   ```

   You can leave out the <protocol value> to display all supported protocols. Sample command and sample output are:

   ```
   CLI> conf hardware pstn 2 1 protocol
   missing option - expected

      <integer>          1   - 4ESS
      <integer>          2   - 5ESS
      <integer>          3   - CAS
      <integer>          4   - DMS
      <integer>          5   - E1CC
      <integer>          6   - NET5
      <integer>          7   - NI2
      <integer>          8   - NTT
      <integer>          9   - QSIGE1
      <integer>          10  - QSIGT1
      <integer>          11  - R2MF
      <integer>          12  - T1CC
      <integer>          13  - DASS2
      <integer>          14  - DPNSS
   ```

Note: R2MF is not supported on the DNI2410TEPEHMP Board. You may assign the same protocol or different protocols to each trunk on the DNI2410TEPEHMP Board, with some limitations. See Guidelines for Assigning Protocols on DNI2410TEPEHMP Boards below.

The following example sets the protocol for device 2 trunk 1 to QSIGE1 (9):

```
CLI> conf hardware pstn 2 1 protocol 9
updated
```

The updated message is displayed.

5. Verify the new settings using the **show** command.

   ```
   CLI> show hardware pstn
   ```

   Information about device ID, trunk ID, interface ID, protocol, and more is displayed in two columns: User Defined Table and Resolved Table. The protocol value will show QSIGE1 (the value “9” is translated) in the User Defined Table and the previous protocol setting for that trunk in the Resolved Table. The user defined value takes effect after the setting has been applied (Step 6),

6. Apply the new settings using the **conf** command. This step validates that the protocol is supported on this board. The setting takes effect on the next media services restart.

   ```
   CLI> conf system pstn apply
   updated
   ```

7. Verify that the User Defined Table and Resolved Table values match, indicating that the new value has been applied.

   ```
   CLI> show hardware pstn
   ```

   Information about device ID, trunk ID, interface ID, protocol, and more is displayed in two columns: User Defined Table and Resolved Table. In this example, after you have restarted media services, the protocol value will show QSIGE1 (the value “9” is translated) in the User Defined Table and in the Resolved Table. Any errors are displayed in the Error Description for each device.

8. Repeat steps 4 through 7 for each PSTN network interface setting to be modified.

Continue with any additional configuration procedures that are applicable to your system. When you are satisfied with all configuration information, restart the media services. See “Restarting Media Services” procedure in the *Dialogic® System Configuration Guide*.

Guidelines for Assigning Protocols on DNI2410TEPEHMP Boards

The protocols supported on the DNI2410TEPEHMP Board are as follows (the protocols are shown as “Group 1” and “Group 2”, which is explained below):

Group 1 Protocol Values	Group 2 Protocol Values
4ESS (T1)	DPNSS (E1)
5ESS (T1)	DASS2 (E1)
CAS (T1)	
DMS (T1)	
NI2 (T1)	
NTT (T1)	
QSIGT1 (T1)	
T1CC (T1)	
E1CC (E1)	
NET5 (E1)	
QSIGE1 (E1)	

You may assign the same protocol or different protocols to each trunk on the board, but all of the protocols on trunks 1-4 must belong to the same group (Group 1 or Group 2) and have the same line type (T1 or E1), and all of the protocols on trunks 5-8 must belong to the same group and have the same line type.

The FCD and PCD file names are updated appropriately after you configure the protocols for the PSTN network interface. See Section 1.38.4.3, “FCD/PCD File Names”, on page 89.

1.38.4.2 Trunk Configuration Using SNMP

The “Configuration Procedures Using SNMP” chapter in the *Dialogic® System Configuration Guide* includes a procedure for configuring PSTN network interfaces. You can follow that procedure for the DNI2410TEPEHMP Board. The procedure is repeated here, with supplementary information for the DNI2410TEPEHMP Board.

1. Verify that the media services and boards are stopped. See “Stopping Media Services” procedure in the *Dialogic® System Configuration Guide*.

2. In the **ipmsHwDM3Trunk** MIB, modify a current PSTN network interface setting for a specific device and network interface (trunk). For example, modify the protocol using **ipmsctHwDM3TrunkProtocol** object. The change takes effect on the next media services restart.

   Note: R2MF is not supported on the DNI2410TEPEHMP Board. You may assign the same protocol or different protocols to each trunk on the DNI2410TEPEHMP Board, with some limitations. See Guidelines for Assigning Protocols on DNI2410TEPEHMP Boards above.

Continue with any additional configuration procedures that are applicable to your system. When you are satisfied with all configuration information, restart the media services. See “Restarting Media Services” procedure in the *Dialogic® System Configuration Guide*.

1.38.4.3 FCD/PCD File Names

The FCD and PCD files make up the configuration file set that is downloaded to the board. The default FCD and PCD files for the DNI2410TEPEHMP Board are *hmposbe_default.fcd* and *hmposbe_default.pcd*.

After you select the protocols by using either CLI or SNMP as shown above, new FCD and PCD files are automatically generated to reflect the protocols that were selected. FCD and PCD file names that begin with "g" are files that have been generated. For example, *gnetworkonly_hmposbe_1_net5_7_qsige1.pcd* is a generated PCD file for a DNI2410TEPEHMP Board using NET5 protocol on one trunk and QSIGE1 protocol on the other seven trunks.

When using SNMP, you can see the FCD/PCD file names in the **ipmsHwDM3Dev** MIB.

When using CLI, you can see the FCD/PCD file names by entering **show hardware dm3**. In the following example, Device ID 1 is the HMP Software virtual device and Device ID 2 is the DNI2410TEPEHMP Board. Sample command and sample output are:

```
CLI> show hardware dm3
Device ID 1
       Logical ID:    1
       Model Number:  32809
       Media Load:    HMP Load
       PCD File Name: 0r360v0e0c0s0f0i0m0a0u0n0g_hib_pur.pcd
       FCD File Name: 0r360v0e0c0s0f0i0m0a0u0n0g_hib_pur.fcd
       PCM Encoding:  Mu-Law
       Driver State:  DOWNLOADED
       CFG ID:        21

Device ID 2
       Logical ID:    2
       Model Number:  7440
       Media Load:    Network Only
       PCD File Name: gnetworkonly_hmposbe_8_net5.pcd
       FCD File Name: gnetworkonly_hmposbe_8_net5.fcd
       PCM Encoding:  A-Law
       Driver State:  DOWNLOADED
       CFG ID:        2
```

Note: It is recommended that you do **not** modify the FCD/PCD file names via the CLI **conf hardware dm3** command or via SNMP. The FCD/PCD file name values are updated when you configure the protocol for the network interface.

For detailed information about FCD and PCD files, see the *Dialogic® System Configuration Guide*.

1.38.5 Runtime Configuration of Echo Cancellation

Refer to Section 1.39, "Enabling Echo Cancellation with Non-Linear Processing", on page 90, for information about runtime configuration of echo cancellation.

1.39 Enabling Echo Cancellation with Non-Linear Processing

With this Service Update 154, the application has the ability to enable or disable echo cancellation with Non-Linear Processing (NLP) on a per channel basis using the Dialogic® Global Call API.

Note: This feature is currently supported on the Dialogic® DNI601TEPHMP, DNI2410TEPEHMP, DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards. Refer to Section 1.29, "Support for Dialogic® DNI/310TEPEHMP, DNI/610TEPEHMP, and DNI/1210TEPEHMP Digital Network Interface Boards", on page 75 and Section 1.38, "Support for Dialogic® DNI2410TEPEHMP Digital Network Interface Board", on page 83 for additional board information.

1.39.1 Feature Description

This feature allows the application to dynamically set echo cancellation parameters, such as NLP and enabled/disabled, at run time. The application uses the existing Global Call APIs **gc_SetConfigData( )** to set (enable/disable) a PSTN channel's echo canceller and **gc_GetConfigData( )** to query its current value for a given line device (channel basis).

New Parameter ID and Values

The following definitions are used for enabling and disabling echo cancellation:

```
Set ID: CCSET_DM3FW_PARM
Parm Id: CCPARM_ECHOCANCEL
Valid Values: CCDM3FW_PARMECHOCANCEL_DISABLE 0x0
              CCDM3FW_PARMECHOCANCEL_ENABLE 0x1
                CCDM3FW_PARMECHOCANCEL_ENABLE_NLP 0x2
```

The target type is GCTGT_CCLIB_CHAN, where the target value is the linedev handle.

Using the CCSET_DM3FW_PARM set ID and the new echo cancellation defines, the application can set up the PSTN channels with echo cancellation capability. These echo cancellation values can be combined as a bitmask to create the following modes of echo cancellation.

Parameter ID	Value	Description
CCDM3FW_PARMECHOCANCEL_DISABLE	0x0	No echo cancellation
CCDM3FW_PARMECHOCANCEL_ENABLE	0x1	Echo cancellation enabled
CCDM3FW_PARMECHOCANCEL_ENABLE | CCDM3FW_PARMECHOCANCEL_ENABLE_NLP	0x3	Echo cancellation enabled, NLP enabled

Note: The following values are not supported and will return an error if used:

```
CCDM3FW_PARMECHOCANCEL_ENABLE_NLP
CCDM3FW_PARMECHOCANCEL_DISABLE|CCDM3FW_PARMECHOCANCEL_ENABLE_NLP
```

Example

```
...
GC_PARM_BLKP echo_blkp = NULL;
int req_id;
...


/* insert parm by value */
if ( gc_util_insert_parm_val( &echo_blkp, CCSET_DM3FW_PARM, CCPARM_ECHOCANCEL, sizeof( char ),
(char)(CCDM3FW_PARMECHOCANCEL_ENABLE | CCDM3FW_PARMECHOCANCEL_ENABLE_NLP)) != GC_SUCCESS )
{

     sprintf(str, "gc_util_insert_parm_val(CCSET_DM3FW_PARM, CCPARM_ECHOCANCEL, sizeof( char ),
     (char)CCDM3FW_PARMECHOCANCEL_ENABLE) Failed");
      printandlog(index, GC_APIERR, NULL, str, 0);
      exitdemo(1);
}

/* Enable Echo Cancellation */
if (gc_SetConfigData(GCTGT_CCLIB_CHAN, port[index].ldev, echo_blkp, 0, GCUPDATE_IMMEDIATE,
&req_id, EV_ASYNC) != GC_SUCCESS) {
     sprintf(str, "gc_SetConfigData(GCTGT_CCLIB_CHAN, targetID:0x%x, mode:EV_ASYNC Failed",
     port[index].ldev);
     printandlog(index, GC_APIERR, NULL, str, 0);
     exitdemo(1);
}
/* delete parm blk */
...
```

1.40 Support for SS7 Functionality

With the Service Update 139, Dialogic® HMP Software 3.1LIN supports Signaling System 7 (SS7) via Signaling Interface Units (SIU) and SIGTRAN (IETF SS7 Signaling over IP). The HMP system provides SIU/SIGTRAN detection, initialization and configuration support. Global Call supports the development of call control applications that use SS7 technology. For more information about SS7 technology, refer to the *Dialogic® Global Call SS7 Technology Guide*.

Release Issues 2

The table below lists issues that can affect the hardware and software supported in the Dialogic® Host Media Processing Software Release 3.1LIN. The following information is provided for each issue:

Issue Type
: This classifies the type of release issue based on its effect on users and its disposition:
  - Known – A minor hardware or software issue. This category includes interoperability issues and compatibility issues. Known issues are still open but may or may not be fixed in the future.
  - Known (permanent) – A known hardware or software issue or limitation that is not intended to be fixed in the future.
  - Resolved – A hardware or software issue that was resolved (usually either fixed or documented) in this release.

Defect No.
: A unique identification number that is used to track each issue reported via a formal Change Control System. Additional information on defects may be available via the Defect Query tool at *http://membersresource.dialogic.com/defects/*.
Note that when you select this link, you will be asked to either LOGIN or JOIN.

SU No.
: For defects that were resolved in a Service Update, the Service Update number is shown. For defects that were resolved when the base release was generally available (before any Service Updates), a "--" is shown. For non-resolved issues, this information is left blank.

Product or Component
: The product or component to which the problem relates, for example, an API.

Description
: A summary description of the issue. For non-resolved issues, a workaround is included when available.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00092285	244	Codec	The decoder occasionally fails to decode MPEG4 frames from the 3G endpoint, resulting in poor video quality.
Resolved	IPY00092278	244	Codec	HMP software does not consider available MPEG4 in-band DCI and instead relies on a user-provided value even though it is invalid and causing high CPU usage.
Resolved	IPY00092200	244	Codec	H263_1998 did not work despite being a valid transcode option with DML framework.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00092084	244	Codec	Play video transcoding fails with the receiving side missing all audio data.
Resolved	IPY00091982	244	Configuration	A TDX_RECORD completion event is not received after calling the **dx_stopch( )** function to terminate record activity.
Resolved	IPY00091502	244	Configuration	The use of the NCM API to restart the Dialogic® Service results in a potential hang during service shutdown.
Resolved	IPY00092243	244	CSP	CSP demo functionality fails on Red Hat Enterprise 5.x systems with the GCC 4.x compiler.
Resolved	IPY00091419	244	Device Management	Calling the dev_PortDisconnect( ) function stops two connections (RTP forking) when transcoding is enabled.
Resolved	IPY00092209	244	Fax	Inbound faxes are received with some of the pages cut off. The missing part appears blank when displayed with an image viewer.
Resolved	IPY00091910	244	Firmware	A "click" is heard when a party is added to a CNF conference.
Resolved	IPY00091783	244	Firmware	A firmware crash results after setting the Signal Detector minimum energy parameter (0x070b).
Resolved	IPY00091867	244	Global Call (IP)	A GCEV_ACCEPT_MODIFY_CALL was incorrectly sent to the application in a supervised transfer scenario where the media was disconnected while the call was moving into a disconnected state.
Resolved	IPY00047881	244	Global Call IP	Applications compiled with an older set of HMP header files fails on gc_Start( ) when IP_VIRTBOARD is used.
Resolved	IPY00091103	244	Global Call IP (SIP)	The application receives two termination events, GCEV_TASKFAIL and GCEV_DISCONNECTED, after calling the **gc_MakeCall( )** function.
Resolved	IPY00055887	244	Global Call IP (SIP)	The application receives a GCEV_TASKFAIL event if the remote end sends an empty ReINVITE during a fax session.
Resolved	IPY00091959	244	Installation	During dlstop on a SUSE Linux Enterprise Server (SLES) 11 system, the DM3 script looks for logging entry /etc/syslog.conf file but the file does not exist on SLES 11 systems.
Resolved	IPY00092395	244	IP Media	Dialogic® HMP software transmits bad AMR multiframe packets when an AMR multi-frame session is preceeded by a session configured for another codec.
Resolved	IPY00092280	244	IP Media	Modify media tests result in a TASKFAIL.
Resolved	IPY00081808	244	IP Media Session Control RTP	The RTP video buffer size for the IP Media device is preset to 1500 bytes, but should be higher for fragmented packets.
Resolved	IPY00091918	244	Licensing	The DM_GetDongleIDs library function fails due to memory-related issues.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00055853	244	Licensing	When generating an additive license file, the additive license generator places the SN keyword on the second to last line. The SN keyword must be on the last line.
Resolved	IPY00081118	244	MSML	The MSML server does not allow calls to join a conference and instead reports an internal server error.
Resolved	IPY00054971	244	MSML	The MSML server fails to read or write media files from remote drives or shares.
Resolved	IPY00092357	244	PSTN Call Control	The ISDN host library does not wait for LineAdmin utility to complete initialization and instead assumes the link is down.
Resolved	IPY00056036	244	SIP Call Control	SDP information is missing from the second INVITE after the stack authenticates the resend of the initial INVITE.
Resolved	IPY00056033	244	SIP Call Control	No 48x response to a SIP INVITE message is received after the licensed channel limit is reached.
Resolved	IPY00056010	244	SIP Call Control	When PRACK is sent to the Dialogic® HMP software with the RAck header missing the CSeq header, it incorrectly retransmits 183 and upon timeout sends 500 internal server.
Resolved	IPY00055976	244	SIP Call Control	When the MIME feature is turned on, the wrong INVITE message is sent after CANCEL within same TCP connection. As a result, Dialogic® HMP Software rejects incoming calls with "getFreeBuffer: no buffer available."
Resolved	IPY00090816	240	Conferencing	Issues occur with RTF logging in the User Mode Bridge Controller (UMBC).
Resolved	IPY00082301	240	CSP	During a CSP load test, **ec_reciottdata( )** was set to terminate in as many as 120 different lengths for MAX_SIL termination conditions. This caused the tone templates to be exceeded which caused the function to fail. ***Workaround:*** The following parameters can be added to the *Hmp.Uconfig* file to adjust the number of Tone Templates and Defines. [sigDet] SetParm=0x0700,128 ! SD_ParmMaxTnTmplts (default=128) SetParm=0x0701,128 ! SD_ParmMaxTnDefs (default=128)
Resolved	IPY00091383	240	Dialogic® HMP Software	Dialogic® HMP software does not operate as expected after the operating system time is modified.
Resolved	IPY00091140	240	Fax	Improper error handling - HMP fax stops processing faxes when it receives a BAD FCS HDLC message after a TSI and before the DCS frame.
Resolved	IPY00091139	240	Fax	HMP fax send/receive failures occur causing all fax channels to be busy.
Resolved	IPY00091138	240	Fax	The fax application locks when sending a fax.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00091093	240	Fax	A fax termination issue occurs while using T.30 FAX with ECM enabled (error correction mode) on HMP.
Resolved	IPY00090932	240	Fax	The customer implemented Fax on Demand to block incoming faxes, but the HMP application receives a call and transmits a fax.
Resolved	IPY00082125	240	Fax	The fax server freezes sporadically and the HMP fax application stops responding. Only a system reboot temporarily resolves the problem.
Resolved	IPY00080125	240	Fax	A race condition in T.38 fax caused send and receive errors.
Resolved	IPY00080124	240	Fax	When the application receives more than 100 pages of fax, the page number restarts from zero (0).
Resolved	IPY00082294	240	Firmware	The application runs for a while and then stops receiving calls and other API events.
Resolved	IPY00081685	240	Firmware	A sporadic firmware crash occurs after a **dx_play( )** function call completes on a voice channel and the same channel is listening to a TDM time slot using a **dx_listen( )** function call.
Resolved	IPY00091023	240	Global Call	The **gc_DropCall( )** function fails and causes a hung port.
Resolved	IPY00091361	240	Global Call IP (SIP)	An application error (offset 000d959, librtfmt.dll) occurs.
Resolved	IPY00091348	240	Global Call IP (SIP)	Dialogic® HMP software rejects an incoming INVITE with "603 Decline". This occurs when the application issues the API functions **gc_OpenEx( )**, **gc_Close( )**, **gc_WaitCall( )**, and **gc_ResetLineDev( )** in particular order.
Resolved	IPY00091292	240	Global Call IP (SIP)	SIP UPDATE method is not included in the Allow header (unless Session Timers are enabled). There appears to be a missing check for UPDATE enabled mask when setting UPDATE in ALLOW header.
Resolved	IPY00091162	240	Global Call IP (SIP)	When a 200 OK response is received for an UPDATE message, the GCEV_EXTENSION event is blindly sent to the application.
Resolved	IPY00091091	240	Global Call IP (SIP)	When **gc_MakeCall( )** is issued with timeout value =0, and one of the SIP stack timer "SIP_STACK_CFG.provisionalTimer" expires, HMP does not send the SIP CANCEL message. As a result, the call cannot be cleared at far end.
Resolved	IPY00090750	240	Global Call IP (SIP)	No Response is sent to invalid INVITE.
Resolved	IPY00090646	240	Global Call IP (SIP)	In 1PCC mode, the SDP answer differs for 18x and 200_OK.
Resolved	IPY00090645	240	Global Call IP (SIP)	Dialogic® HMP Software sip stack incorrectly sends a “486 Busy Here” message in response to SIP PRACK message received.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00081732	240	Global Call IP (SIP)	A GCEV_REQ_MODIFY_CALL event is reported while a previous **gc_AcceptModifyCall( )** is in progress.
Resolved	IPY00055507	240	Global Call IP (SIP)	In 1PCC mode, SIP ReINVITE Codec Reporting appears to be wrong or corrupted.
Resolved	IPY00055506	240	Global Call IP (SIP)	HMP transfer disconnects after 200OK is received for NOTIFY when attempting to make an outbound transfer using NOTIFY messages to track the status to the transferring party.
Resolved	IPY00055439	240	Global Call IP (SIP)	Dialogic® HMP Software cannot issue a re-INVITE on a channel after rejecting a previous remote re-INVITE.
Resolved	IPY00091529	240	Installation	Files specific to the AdvancedTCA release still appear in the DATA directory (/usr/dialogic/data).
Resolved	IPY00091132	240	IP Media	Audio transcoding between G.711 to G.729 is not successful.
Resolved	IPY00055625	240	IP Media	Outbound 1PCC SIP call connects as G729AB but reports the receive GCEV_EXTENSION as G729A.
Resolved	IPY00082344	240	PSTN Call Control	The gc_GetCallInfo( ) function returns a redirecting number when none is present in the incoming IAM message.
Resolved	IPY00090994	240	Voice	The **dt_listen( )** function locks. A mutex was used in the waiting message queue instead of a condition variable.
Resolved	IPY00079689	240	Voice	A DNI transaction record issue causes the **dx_mreciottdata( )** function to not operate as expected.
Resolved	IPY00081676	237	Device Management	The **dev_GetResourceReservationInfo( )** function returns two same DMEV_GET_RESOURCE_ RESERVATIONINFO events.
Resolved	IPY00082163	237	Firmware	The **mm_Play( )** function returns a play failure for 3gp play of helix generated files.
Resolved	IPY00081664	237	Firmware	Firmware crashes when the input JPEG or YUV still image file is too large to be processed by the firmware.
Resolved	IPY00082165	237	Global Call IP (SIP)	Dialogic® HMP software failed to generate multiple PRACK responses.
Resolved	IPY00055290	237	Global Call IP (SIP)	When the application tries to de-register binding, which was previously registered with a particular IP-PBX, the **gc_ReqService( )** function returns success, but the GCEV_SERVICERESP event indicates error.
Resolved	IPY00082088	237	IP Media	The application does not receive GCEV_ALARM events (LAN DISCONNECT) for IPT devices, when DTI devices are opened before IPT devices during the initialization process.
Resolved	IPY00080472	237	IP Media	An issue with the RV stack causes incorrect handling of unexpected synchronous messages.
Resolved	IPY00081235	237	Licensing	License activation fails when using ethernet ports. Any subsequent attempt to start services fails with an "invalid host id " error message.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00081399	233	3G-324M	A segmentation fault occurs with video transcoding.
Resolved	IPY00081661	233	Device Management	A call to the **dev_PortDisconnect( )** function results in the application crashing as segmentation fault.
Resolved	IPY00081284	233	Global Call IP (SIP)	Session Timer settings appear to be reset upon a re-INVITE to the Dialogic® HMP software.
Resolved	IPY00081264	233	Global Call IP (SIP)	UAS Session Timer programming does not survive the application's re-INVITE request.
Resolved	IPY00081132	233	Global Call IP (SIP)	The SIP hold/retrieve call scenario using the gc_ReqModifyCall( ) and gc_AcceptModifyCall( ) functions does not perform as expected.
Resolved	IPY00055020	233	Global Call IP (SIP)	Proxy-Authentication for registration and re-registration encounters problems with multi-user registrations.
Resolved	IPY00054970	233	Global Call IP (SIP)	Dialogic® HMP software responds incorrectly upon receipt of a 488 message from the UAS when issuing a Session Timer re-INVITE or UPDATE message. The re-INVITE 488 message response causes the session to drop immediately instead of the Session Timer functionality dropping the session.
Resolved	IPY00054969	233	Global Call IP (SIP)	A Call Transfer issue causes REFER to fail after receipt of an INVITE.
Resolved	IPY00081665	233	IP Media	The application is unable to set a custom port in the REGISTRAR ADDRESS when using proxy bypass because the port number is not extracted from the requested URI.
Resolved	IPY00080727	233	IP Media	Different QoS alarms default parameter values differ from user docuementation. (See documentation update for the Dialogic® IP Media Library API Programming Guide and Library Reference.)
Resolved	IPY00081571	233	IP Media Session Control RTP	RTP alarm events are missing when Dialogic® HMP acts as sender without a valid receiver.
Resolved	IPY00081489	233	IP Media Session Control RTP	The RTP alarm is on for approximately two minutes and then automatically resets when there is no active stream.
Resolved	IPY00081756	233	Multimedia	No MMEV_PLAY event was received when an audio/video file played out after calling the **mm_Play( )** function via streaming.
Resolved	IPY00081410	233	Voice	The application stops receiving DX events and the I/O functions fail.
Resolved	IPY00081126	229	3G-324M	The **m3g_ModifyMedia( )** function fails when the wrong ports are connected.
Resolved	IPY00081254	229	Firmware	A firmware crash occurs when the **dev_PortConnect( )** function is issued before the **m3g_ModifyMedia( )** function.
Resolved	IPY00081277	229	Installation	The Uninstall process does not remove all RPM Package Managers (RPMs).

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00081211	229	Multimedia	An **mm_Play( )** function call without MM_RUNTIME_CONTROL terminates based on a previous MM_RUNTIME_CONTROL.
Resolved	IPY00081107	229	Multimedia	Under some conditions, a small amount of video is recorded before the first I-Frame is detected during a multimedia record.
Known (permanent)	IPY00081277	227	Installation	Service Update 227 Uninstall process does not remove all RPM Package Managers (RPMs). When downgrading from SU 227 to an earlier build (for example, uninstall SU 227 and install SU 226), the installation of the earlier build (SU 226) fails. ***Workaround:*** After uninstalling build 227 and prior to installing SU 226 or earlier builds, you should first manually remove the RPMs that were not removed during the SU 227 Uninstall process. Here is the procedure: 1. Uninstall SU 227. 2. Log out and then log in. 3. Query the dialogic RPMs that are not removed from the SU 227 Uninstall process as follows: “rpm –qa | grep lsb-dialogic” For example on a gcc3.4.3 system, [root@pylnxlab9702 ~]# rpm -qa | grep lsb-dialogic lsb-dialogic-hmp-sdk -3.1.0.0-283 lsb-dialogic-hmp-lic-3.1.0.0-225_gcc3.4 4. Manually remove the above RPM as follows: [root@pylnxlab9702 ~]# rpm -e lsb-dialogic-hmp-sdk-3.1.0.0-283 [root@pylnxlab9702 ~]# rpm -e lsb -dialogic-hmp-lic-3.1.0.0-225_gcc3.4 5. Install the desired HMP 3.1LIN Service Update (for example, SU 226).
Resolved	IPY00080790	227	Multimedia	A larger than expected time gap may occur between RTP packets sent during a multimedia play from memory or stream..
Resolved	IPY00080658	227	Multimedia	A ghost image sometimes appears after stopping a video play and then starting another one.
Resolved	IPY00080471	226	3G-324M	Firmware is removing one digit for E71 if the same digit is pressed in quick succession.
Resolved	IPY00080364	226	3G-324M	The firmware crashes with the error “Audio Stack Buf Pattern does not match mem corruption”.
Resolved	IPY00079897	226	3G-324M	The Gateway application stops receiving UII digits after detecting digits when using an HTC hand phone.
Resolved	IPY00079787	226	3G-324M	The second call to **m3g_Close( )** on a 3G-324M device fails.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00079742	226	3G-324M	When acting as H.245 MASTER, the platform uses a rejection cause indicating unsuitable reverse parameters when receiving a bidirectional video OLC (without reverse Parameters of nullData).
Resolved	IPY00079734	226	3G-324M	Firmware crashes while parsing remote terminal alternate capability.
Resolved	IPY00080622	226	Conferencing (DCB)	The conferencing application crashes during load testing.
Resolved	IPY00079768	226	DTMF	DTMF digit collection fails when using a Dialogic® DNI board.
Resolved	IPY00079931	226	Fax	Inbound faxes are missing detection of pages and TCF signals.
Resolved	IPY00079435	226	Fax	A high percentage of inbound faxes are received as blank or partly cut-off.
Resolved	IPY00079346	226	Fax	After calling the **fx_stopch( )** function at phase C of the fax receive side, there is no TFX_FAXERROR event reported nor is the state of fax channel IDLE.
Resolved	IPY00079344	226	Fax	Dialogic® HMP software stores received faxes with a non-metric resolution calibration.
Resolved	IPY00080324	226	IP Media	Dialogic® HMP software fails to send RTP audio in a hair-pinned NbUP to IP connection.
Resolved	IPY00080130	226	IP Media	Dialogic® HMP software may stop processing packets when the incoming packets have UDP errors
Resolved	IPY00079902	226	IP Media	Recordings contain distorted voice when incoming RTP packets contain a header extension.
Resolved	IPY00079375	226	IP Media	Updated RTCP Parser to add JB High Watermark and JB Low Watermark fields.
Resolved	IPY00080556	226	MSML	Dialogic® HMP software does not send all audio and video RTP packets.
Resolved	IPY00080564	226	Multimedia	An MMEV_PLAY event is never received when playing a corrupt video file.
Resolved	IPY00080336	226	Multimedia	A ghost image sometimes appears after stopping a video play and then starting another one.
Resolved	IPY00080297	226	Multimedia	Added EOD event to multimedia streaming API.
Resolved	IPY00080129	226	Multimedia	Dialogic® HMP software sometimes closes an application's file descriptor during a multimedia stream or memory record.
Resolved	IPY00079981	226	Multimedia	Video recordings generated by a multimedia stream or memory record are sometimes corrupted.
Resolved	IPY00079929	226	OA&M	The CLI service fails to start when IPv6 is disabled.
Resolved	IPY00079866	226	PSTN Call Control	No response is sent back to the application upon receipt of a user-to-user service 1 or 2 request.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00079825	226	PSTN Call Control	When receiving an IAM with the continuity check indicator set to spare (illegal value), the application handles it as if a "continuity check required" indicator was received.
Resolved	IPY00080570	226	Voice	Dialogic® HMP software crashes when a license without RTP is used, e.g., a voice only license.
Resolved	IPY00079137	223	3G-324M	The **m3g_Stop( )** function throws an unhandled exception when the libm3g library is dynamically linked.
Resolved	IPY00078369	223	3G-324M	An audio problem occurs with a 3G call to a soft phone.
Resolved	IPY00078344	223	3G-324M	Interoperability problems occur with the adaptation layer.
Resolved	IPY00045089	223	3G-324M	An incompatibility with a 3G-324m service endpoint causes missing DTMFs or no video.
Resolved	IPY00046378	223	Conferencing	A segmentation fault occurs in the CNF module when the application terminates.
Resolved	IPY00078309	223	Configuration	A memory leak occurs when the status changes multiple times continuously.
Resolved	IPY00045524	223	Device Management	The **dev_connect( )** function fails when used between M3G and DNI devices.
Resolved	IPY00079530	223	Global Call IP (SIP)	The application is unable to place a call on hold and then retrieve it when using G.723.
Resolved	IPY00079518	223	Global Call IP (SIP)	After receiving the GCEV_TASKFAIL event, the **gc_ResetLineDev( )** function does not return the GCEV_RESETLINEDEV event.
Resolved	IPY00079509	223	Global Call IP (SIP)	The Min-SE header is present in the outgoing INVITE even though session timers on the channel are disabled.
Resolved	IPY00079483	223	Global Call IP (SIP)	Cannot initiate sendOnly or recvOnly calls using parameter IP_CAP_DIR_LCLSENDONLY or IP_CAP_DIR_LCLRECVONLY.
Resolved	IPY00079466	223	Global Call IP (SIP)	The supported 'replaces' header has a [space] before carriage return line feed (CRLF) causing an ingress call to fail.
Resolved	IPY00079393	223	Global Call IP (SIP)	The SIP Allow header is omitted in response messages to inbound calls.
Resolved	IPY00079374	223	Global Call IP (SIP)	Dialogic® HMP software does not obtain the origination point in multi-step diversions.
Resolved	IPY00079251	223	Global Call IP (SIP)	The SIP Allow header does not include the 'INFO' method.
Resolved	IPY00079108	223	Global Call IP (SIP)	IPPARM_OFFERED_FASTSTART_CODER/ GCSET_CHAN_CAPABILITY reporting for SIP G723.1 is always chosen at the default bit rate of 6.3k if remote side does not specify the bit rate.
Resolved	IPY00078440	223	Installation	A continuous Query ipmsLicenseFileSelected and ipmsLicenseExpirationDate in the Dialogic® HMP software causes snmpwalk to hang.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00079186	223	MSML	The MSML media server deadlocks when dialog exists at the same time a dialog end request is received.
Resolved	IPY00079639	223	Multimedia	Blank video results when the **mm_play( )** function is called for audio and video after the function was called for audio only.
Resolved	IPY00079551	223	PSTN	IAM messages exceed the maximum allowed.
Resolved	IPY00079502	223	Voice	An RTP timestamp reset causes one-way audio.
Resolved	IPY00079201	223	Voice	When using Dialogic® DNI boards, the **dx_mreciottdata( )** function does not operate properly.
Resolved	IPY00078397	221	Dialogic® HMP software	An application may crash with the following message: DlgcHost_Utility:SystemException.
Resolved	IPY00079506	221	Dialogic® HMP software	The dlstart command intermittently fails during media startup.
Resolved	IPY00078800	221	Documentation	Updates were made to the Dialogic® Device Management API Library Reference available on the documentation bookshelf.
Resolved	IPY00044997	221	Documentation	Updates were made to the Dialogic® IP Media API Programming Guide and Library Reference available on the documentation bookshelf.
Resolved	IPY00044586	221	Documentation	Updates were made to the Dialogic® IP Media API Programming Guide and Library Reference available on the documentation bookshelf.
Resolved	IPY00043498	221	Documentation	Updates were made to Section 9.1.2 of the Dialogic® System Configuration Guide available on the documentation bookshelf.
Resolved	IPY00042207	221	Documentation	Updates were made to Sections 9.1.1 and 9.1.2 of the Dialogic® System Configuration Guide available on the documentation bookshelf.
Resolved	IPY00044901	221	Fax	Fax failures occur intermittently with certain fax machines.
Resolved	IPY00078440	221	Installation	A continuous Query ipmsLicenseFileSelected and ipmsLicenseExpirationDate in the Dialogic® HMP software causes snmpwalk to hang.
Resolved	IPY00078438	221	Installation	Dialogic® boards will not download due to the following error: SUT-857 kernel: supp_process_receive: maxbind = 100.
Resolved	IPY00079023	221	IP Media	A wrong return results when using the **ipm_GetSessionInfo( )** function to get RTCP and QoS stats.
Resolved	IPY00078793	221	IP Media	RTCP Receiver Reports may not be generated after a sender stops sending data.
Resolved	IPY00078792	221	IP Media	An RTCP Receiver Report is not generated periodically on an idle IP Media session.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00078791	221	IP Media	After a Sender becomes a Receiver in the same IP Media session, an embedded RTCP Sender Report may be generated instead of an RTCP Receiver report.
Resolved	IPY00079132	221	Voice	The **dx_getdig( )** function fails after a conference member is muted.
Resolved	IPY00078474	221	Voice	The application does not receive a TDX_RECORD event after the **dx_reciottdata( )** function is issued followed by the **dx_stopch( )** function on single voice channel.
Resolved	IPY00078369	210	3G-324M	Audio problem is observed when calling from 3G phone to softphone.
Resolved	IPY00045033	210	IP Media	Incorrect codec #defines in ipmlib.h.
Resolved	IPY00045346	210	SS7	When running SS7, COT tests periodically fails under load.
Resolved	IPY00078392	210	Voice	When using IO_STREAM, the second **dx_playiottdata( )** on a channel immediately gets TDX_PLAY with EOD.
Resolved	IPY00045407	210	Voice	When using the unsupported coder CODER_TYPE_NONSTANDARD in **ipm_StartMedia( )**, a segmentation fault may occur.
Resolved	IPY00045077	210	Voice	When recording is terminated using DTMF, the last few seconds of audio is sometimes lost and not recorded.
Resolved	IPY00044834	204	3G-324M	When M3G trace with logfile is enabled, this may cause a system failure if the logfile reaches over 2GB.
Resolved	IPY00045006	204	Fax	Exception in the fax component causes resources to be stuck in non-idle state.
Resolved	IPY00044978	204	Fax	The **fx_rcvfax( )** function fails with a -1 error.
Resolved	IPY00044750	204	Fax	TFX_FAXERROR event is not received after calling **fx_stopch( )**.
Resolved	IPY00045156	204	Firmware	Incorrect timestamps are displayed in the m3g firmware traces.
Resolved	IPY00044285	204	IP Media	During transit from ISDN to VoIP, ipm_Listen() fails with “error=0xa=Synchronization object timeout” message.
Resolved	IPY00044224	204	OA&M	The web agent service fails to start during starting Dialogic® services.
Resolved	IPY00044447	203	3G-324M	**m3g_StopMedia( )** fails when m3g devices connected HDX as Sendonly.
Resolved	IPY00043836	203	3G-324M	**m3g_StopH245( )** fails whenever the application isn’t able to close the LC before calling the function.
Resolved	IPY00044235	203	Conferencing (DCB)	After building the max number of conferences, then building a conference again, the API returns error.
Resolved	IPY00044926	203	Fax	When **fx_stopch( )** is called, no TFX_FAXERROR event is received.
Resolved	IPY00044125	203	Firmware	Crash when running load test; result is SIGSEV from OS.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00043860	203	Firmware	Problem connecting outbound calls with Avaya switch. When HMP software is connected to the Avaya switch, the switch sends an unexpected empty Terminal Capabilities Set (TCS) and an Open Logical Channel (OLC) in the middle of the call. As a result, the state machine does not handle the signaling connect event in the media.
Resolved	IPY00044700	203	Global Call IP	An incorrect processing of certain H.323 parameters in a call disconnect caused a library exception condition that trickled up to the customer application as a crash.
Resolved	IPY00044633	203	Global Call IP (SIP)	After having initialized Outbound proxy SIP port to 7070, setting IP Header “Route: <sip:gw1.dialogic.com:7070;lr>\0”; causes sip port to change to 5060.
Resolved	IPY00044578	203	Global Call IP (SIP)	GCEV_ALERTING event is not generated for an ingress 183 Session Progress response.
Resolved	IPY00044544	203	Global Call IP (SIP)	Core dump in librtf (gc_h3r module) when sending certain contents in SIP diversion field.
Resolved	IPY00044509	203	Global Call IP (SIP)	Various calls to **gc_ReqService( )** fail when attempting to query and delete specific AORs
Resolved	IPY00044488	203	Global Call IP (SIP)	When the CGPN and CDPN optional IE's are not present in an incoming setup message, the H323 stack also does not put anything in.
Resolved	IPY00044273	203	Global Call IP (SIP)	The application cannot issue a reINVITE from audio to T.38 Fax using the **gc_ReqModifyCall( )** function in T38_MANUAL_MODIFY_MODE.
Resolved	IPY00044219	203	Global Call IP (SIP)	Manual transfer ends with a disconnect event.
Resolved	IPY00043399	203	Global Call IP (SIP)	When the Session Timers (MinSE and Session-Expires) were set to zero, the request and response messages contained the “timer” parameter in the Supported header and Min-SE:0.
Resolved	IPY00043940	203	IP Media	Multicast fails unless local and remote coder and IP port info are set when calling start media.
Resolved	IPY00042913	203	RTP	Modify the media stream mode during the active call failed; gc_h3r ERR1 validateCaps failed: [-996].
Resolved	IPY00044118	197	Firmware	After HMP dlstart, there is OAMIPC error and core dump along with a core file generated in /usr/dialogic/data.
Resolved		192	Conferencing	When running conferencing at max density, the application can potentially crash.
Resolved	IPY00043038	192	Device Management	When called in multiple threads, **dev_GetTransmitPortInfo(ASYNC)** does not return an event in one thread.
Resolved	IPY00043133	190	Call Control	When using the show hardware PSTN command to display the CDP file being used, the filename shown is not the configured CDP file.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00043023	190	Configuration	While setting TDM Bus configuration with passive mode in dlgclockdaemon, some parameters are not properly initialized resulting in a core dump.
Resolved	IPY00043214	190	Device Management	When **dev_PortDisconnect( )** fails and returns non-zero, it is not returning correct error code. **ATDV_LASTERR( )** returns zero and **ATDV_ERRMSGP( )** returns "No Error" message.
Resolved	IPY00042917	190	Firmware	ssp_x86Linux_boot exits abnormally.
Resolved	IPY00042933	190	Multimedia	**mm_ErrorInfo( )** does not provide useful info and no sign of call to **mm_ErrorInfo( )** in RTF logs.
Resolved	IPY00043216	190	Voice	When playing multiple voice files, some memory files are missing to play.
Resolved	IPY00042216	187	Configuration	Dialogic® services fail to start if disabling Non Linear Processing (NLP) in the *.config* file.
Resolved	IPY00042716	187	IP Media	The application cannot get the local address of type TLS to set in the header file.
Resolved	IPY00042519	187	Multimedia	Poor audio occurs with multimedia play.
Resolved	IPY00041301	174	Demos	When attempting to run the mmdemo, **mm_Play( )** may fail.
Resolved	IPY00041064	174	DNI Firmware	The D channel backup (DCBU) switchover does not work. The standby D channel is not brought up after the primary D channel goes down.
Resolved	IPY00041542	174	Multimedia	When video data is not received, multimedia recording may fail.
Resolved	IPY00041590	174	Multimedia API	Documentation: The documentation for the **mm_StreamRead( )** function does not explain the return value of the *pDataSize parameter.
Resolved	IPY00041454	174	Multimedia API	Documentation: The BufferMode field description in the MM_STREAM_OPEN_INFO data structure is incorrect.
Resolved	IPY00041320	174	Multimedia API	Documentation: The paragraph for the MMEV_VIDEO_HIGHWATER should not have a second sentence.
Resolved	IPY00041508	174	RTP	When playing 8K linear 16 bit PCM files, **dx_Play( )** does not terminate.
Resolved	IPY00040018	157	DNI	Routing failures may occur when running multiple processes.
Resolved	IPY00039666	157	Fax	Sendfax may not complete when sending a polled fax even though entire fax is sent and received.
Resolved	IPY00041041	157	RTP	RFC2833 is not working properly with RFC2198. RFC2198 payload type was not checked for correctly.
Resolved	IPY00039853	157	RTP	While running at high densities, socket errors may occur with the following error message, "[ERROR RTP OTHER] rtpsocket.c 193."

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00039715	139	Configuration	If a board is disabled, the download will fail. When a physical board is not in use, it must be removed from the system.
Resolved	IPY00039825	139	DNI	When using Dialogic® DNI boards on some servers running SUSE Linux 9.3, it is possible to experience a degradation in streaming performance resulting in poor audio quality and tone detection. This condition is accompanied by the following error in the /var/log/messages file and requires Dialogic® services to be restarted. The message indicating this condition in **/var/log/message** is "ssp_x86Linux_boot: null_cfsp.c: losing data."
Resolved	IPY00039858	139	R4 Application	User cannot run more than 8 simultaneous R4 applications. The opening of device in 9th process will fail. For details on the resolution, please see update in the Dialogic® Host Media Processing Software for Linux Configuration Guide section.
Resolved	IPY00081664	37	Firmware	Firmware crashes when the input JPEG or YUV still image file is too large to be processed by the firmware.
Known	IPY00040483		3G-324M	Received native G.723 audio may not be correctly transcoded with 3G-324M. ***Workaround:*** Use AMR-NB instead of G.723.
Known	IPY00040471		3G-324M	Performing **dx_dial( )** over 3G-324M with linear G.723 may result in some missing digits. ***Workaround:*** Use AMR-NB instead of G.723.
Known (permanent)	IPY00038004		CLI	CLI does not fail gracefully on the fourth login session when three other users are logged in. When one of the sessions frees up, the fourth user will be allowed in.
Known	IPY00037899		CLI	There appears to be sporadic crashes of the CLI at random times. ***Workaround:*** Restart the CLI agent by issuing the "dlservices cliagent start" command and then login to the CLI again.
Known	IPY00039990		DNI Fax	Fax call errors may occur when operating with more than one DNI2410 board in the system.
Known	IPY00041750		Fax	When running both PSTN and T.38 fax on the same system, a high rate of PSTN fax errors may occur.
Known	IPY00039059		Fax	When sending multi-page TIFF and grayscale JPEG in the same transaction, no fax is received.
Known	IPY00038947		Fax	The error message "No response after sending page" appears after sending fax and switching to receive in turnaround polling mode.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Known	IPY00038940		Firmware	PESQ scores for G.729/G.729A are occasionally below the minimum requirement, however the average PESQ scores meet the criteria.
Known			Global Call / 3G324M	An error will be returned for **gc_SetConfigData( )** when setting transparent mode. There is no need to set the transparent mode while running 3G324M applications.
Known	IPY00036641		Licensing	Host-based licenses on machines with more than one ethernet interface card must be locked to the MAC address of “eth0”. ***Workaround:*** If the MAC address of eth0 is unknown, run the “ifconfig” command to obtain the MAC address for eth0. If eth0 is not listed in the output (eth0 is disabled and using eth1 or higher) you may need to setup eth0 with a temporary address (e.g. 192.168.0.1) to obtain its MAC address. Leave the system configured this way. In this case eth0 does not need to be connected to the network.
Known	IPY00044980		Multimedia	For video play and record, Red Hat 5.0 is supported in NFS and RAID configurations only when running over 60 channels on a system with a local hard drive.
Known (permanent)	IPY00040928		R4 Application	A problem occurs when the application is linked with Dialogic® libraries. ***Workaround:*** When compiling an application, you must list Dialogic® libraries **before** all other libraries such as operating system libraries.
Known	IPY00039387		RTF	The RTF printout “not enough space in stream source (overrun)” should appear as a warning instead of error message. This causes no functionality issues and is only in the RTF log.
Known	IPY00041133		RTP	While running higher densities on a SUSE machine with G.723 or G.729 codecs, some DTMF digits may be missed.
Known	IPY00037701		RTP	The warning “encoder: ERROR: ippsLinToMuLaw_16s8u( ) - IppStatus:-6” message appears. The message can be ignored since it does not cause a loss of functionality.
Known	IPY00037489 (IPY00037488)		RTP	The following sequence of operations on an IPM device may result in incorrect audio being transmitted in the RTP stream. dev_connect(IPM_Handle,AA_Handlexxx) dev_disconnect(IPM_Handle,AA_Handlexxx) ipm_listen(IPM_Handle,timeslotxxx) dev_connect(IPM_Handle,AA_Handleyyy)
Known (permanent)	IPY00044550		Voice	The **dx_setchxfercnt( )** function is not supported in Dialogic® HMP Software.

Issues Sorted by Type, Dialogic® Host Media Processing Software Release 3.1LIN (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Known	IPY00039374		Voice	There is discrepancy between the size specified in the iott.io_length and the file size created in the **dx_reciottdata( )** function.
Known (permanent)	IPY00037916		Voice	Call progress positive voice detection may fail half the time when running on AMR coders. Positive voice detection/ PAMD does not work reliably on the following AMR bit rates: 5.9k, 6.7k, 7.4k, 7.95k
Known	IPY00036197		Voice	When using 200 or more channels, users should not use the MAXSIL feature because doing so can cause TDX_RECORD events to be triggered prematurely.

Documentation Updates 3

The documentation updates for Dialogic® Host Media Processing Software Release 3.1LIN are divided into the following sections, which correspond to the top level categories used on the online documentation navigation page:

3.1 System Release Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® Host Media Processing Software Release 3.1LIN Release Guide

3.1.1 Dialogic® Host Media Processing Software Release 3.1LIN Release Guide

In Service Update 246, update to Section 2.2, Basic Software Requirements, Supported Compilers,

Add a bullet to show support for GNU Compiler Collection (GCC) 4.1.1.

Update to Section 2.2, Basic Software Requirements, Supported Operating Systems.

Add a bullet to show support for Red Hat Enterprise Linux Release 5.0 Update 2.

Note: When Red Hat Enterprise Linux Release 5.0 Update 2 is installed, select the following file: *ibstdc++ version 5 libs.*

3.2 Installation and Configuration Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide
- Dialogic® Host Media Processing Software for Linux Configuration Guide
- Dialogic® Global Call CDP Configuration Guide

3.2.1 Dialogic® Host Media Processing Software Release 3.1LIN Software Installation Guide

Update to Section 3.1.2, Obtaining an Evaluation License.

The correct location to obtain an evaluation license is http://www.dialogic.com/products/ip_enabled/download/hmp_download.htm.

3.2.2 Dialogic® Host Media Processing Software for Linux Configuration Guide

In Service Update 244, a new version (05-2519-004) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.This guide was previously named the *Dialogic® System Configuration Guide*.

3.2.3 Dialogic® Global Call CDP Configuration Guide

There are currently no updates to this document.

3.3 OA&M Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® Host Media Processing Diagnostics Guide

3.3.1 Dialogic® Host Media Processing Diagnostics Guide

In Service Update 246, update to Section 7.6.2, “Cleaning the RTF Log File”

Replace `rtftool clean [-f]` description with the following text:

Deletes the RTF log files with names starting with “rtflog-” and containing a “txt” string. When the -f option is used, RTF will not ask the user for confirmation before deleting the log file.

3.4 Programming Libraries Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® 3G-324M API Programming Guide and Library Reference
- Dialogic® Audio Conferencing API Library Reference
- Dialogic® Audio Conferencing API Programming Guide
- Dialogic® Conferencing API Library Reference
- Dialogic® Conferencing API Programming Guide

- Dialogic® Continuous Speech Processing API Library Reference
- Dialogic® Continuous Speech Processing API Programming Guide
- Dialogic® Device Management API Library Reference
- Dialogic® Digital Network Interface Software Reference
- Dialogic® Fax Software Reference
- Dialogic® Global Call API Library Reference
- Dialogic® Global Call API Programming Guide
- Dialogic® Global Call E1/T1 CAS/R2 Technology Guide
- Dialogic® Global Call IP Technology Guide
- Dialogic® Global Call ISDN Technology Guide
- Dialogic® Global Call SS7 Technology Guide
- Dialogic® IP Media Library API Programming Guide and Library Reference
- Dialogic® Multimedia API Programming Guide and Library Reference
- Dialogic® Standard Runtime Library API Library Reference
- Dialogic® Standard Runtime Library API Programming Guide
- Dialogic® Voice API Library Reference
- Dialogic® Voice API Programming Guide

3.4.1 Dialogic® 3G-324M API Programming Guide and Library Reference

There are currently no updates to this document.

Note: A new version (05-2558-005) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.2 Dialogic® Audio Conferencing API Library Reference

There are currently no updates to this document.

3.4.3 Dialogic® Audio Conferencing API Programming Guide

Update to Chapter 11, Building Applications.

Added the following sentence to Section 11.1, Compiling and Linking: When compiling an application, you must list Dialogic libraries before all other libraries such as operating system libraries.

3.4.4 Dialogic® Conferencing API Library Reference

There are currently no updates to this document.

3.4.5 Dialogic® Conferencing API Programming Guide

Update to Chapter 8, Building Applications.

Added the following sentence to Section 8.1, Compiling and Linking: When compiling an application, you must list Dialogic libraries before all other libraries such as operating system libraries.

3.4.6 Dialogic® Continuous Speech Processing API Library Reference

There are currently no updates to this document.

3.4.7 Dialogic® Continuous Speech Processing API Programming Guide

Update to Chapter 8, Building Applications.

Added the following sentence to Section 8.2, Compiling and Linking: When compiling an application, you must list Dialogic libraries before all other libraries such as operating system libraries.

3.4.8 Dialogic® Device Management API Library Reference

There are currently no updates to this document.

Note: A new version (05-2222-010) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.9 Dialogic® Digital Network Interface Software Reference

There are currently no updates to this document.

3.4.10 Dialogic® Fax Software Reference

There are currently no updates to this document.

3.4.11 Dialogic® Global Call API Library Reference

Update to **gc_util_insert_parm_val( )** function (IPY00043078).

In the description for **gc_util_insert_parm_val( )**, a note should be added stating that **gc_Start( )** must be called before **gc_util_insert_parm_val( )**. Also, the code example should be replaced with the following:

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

void main( )
```

```
{
    GC_PARM_BLKP my_blkp = NULL;
    GC_PARM_DATAP my_parmp;
    GC_INFO gc_error_info; /* GlobalCall error information data */
    int type = 1;

    /* Issue a gc_Start() call to initialize the library */
    if ( gc_Start(NULL) != GC_SUCCESS )
    {
        /* process error return as shown */
        gc_ErrorInfo( &gc_error_info );
        printf ("Error: gc_Start(), GC ErrorValue: 0x%hx - %s, CCLibID: %i - %s,
                CC ErrorValue: 0x%lx - %s\n", 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);
    }

    /* insert parm by reference */
    if ( gc_util_insert_parm_ref( &my_blkp, GC_SET_SERVREQ, PARM_REQTYPE,
                                  sizeof( int ), &type ) != GC_SUCCESS )
    {
        /* Process error */
    }
    /* insert parm by value */
    if ( gc_util_insert_parm_val( &my_blkp, GC_SET_SERVREQ, PARM_ACK,
                                  sizeof( short ), GC_ACK ) != GC_SUCCESS )
    {
        /* Process error */
    }

    /* Now we should have a GC_PARM_BLK with 2 parameters */

    /* Following use of gc_util_next_parm retrieves the first parameter in a
    * GC_PARM_BLK, which in this case is PARM_REQTYPE */
    my_parmp = gc_util_next_parm( my_blkp, NULL );

    /* Retrieve the next parameter after getting the first one */
    my_parmp = gc_util_next_parm( my_blkp, my_parmp );

    /* This function finds and returns specified parameter, NULL if not found */
    my_parmp = gc_util_find_parm( my_blkp, GC_SET_SERVREQ, PARM_ACK );

    /* After GC_PARM_BLK is no longer needed, delete the block */
    gc_util_delete_parm_blk( my_blkp );

    /* Set my_blkp to NULL now that the block has been deleted */
    my_blkp = NULL;


    /* Issue gc_Stop() Next */
    if (gc_Stop() != GC_SUCCESS )
    {
        /* process error return as shown */
        gc_ErrorInfo( &gc_error_info );
        printf ("Error: gc_Stop(), GC ErrorValue: 0x%hx - %s, CCLibID: %i - %s,
                CC ErrorValue: 0x%lx - %s\n",
                gc_error_info.gcValue, gc_error_info.gcMsg,
                gc_error_info.ccLibId, gc_error_info.ccLibName,
                gc_error_info.ccValue, gc_error_info.ccMsg);
    }
}
```

3.4.12 Dialogic® Global Call API Programming Guide

Update to Chapter 12, Building Applications.

Added the following sentence to Section 12.1, Compiling and Linking: When compiling an application, you must list Dialogic libraries before all other libraries such as operating system libraries.

3.4.13 Dialogic® Global Call E1/T1 CAS/R2 Technology Guide

There are currently no updates to this document.

Note: A new version (05-2445-003) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.14 Dialogic® Global Call IP Technology Guide

Update to Chapter 4, IP-Specific Operations.

New information is added for SIP Session Timer.

SIP Session Timer

Feature Description

The feature provides a keepalive mechanism for the SIP to determine whether a session is still active using an extension defined in RFC 4028. The session timer uses three new fields in the IP_VIRTBOARD data structure to enable the timer, and to set the values for session expiration and minimum time. For negotiation between the user agent server (UAS) and user agent client (UAC), the session timer uses new SIP IP parameters available via the Global Call API. A Session-Expires header included in the 2xx response of an initial INVITE determines the session duration. Periodic refresh enables extending the duration of the call and allows both User Agents and proxies to determine if the SIP session is still active. The refresh of a SIP session is done through a re-INVITE or UPDATE. The Session-Expires header in the 2xx response also indicates which side will be the refresher; that is, the side responsible for generating the refresh request. The refresher can be the UAC or UAS. The refresher should generate a refresh request (using re-INVITE or UPDATE) before the session expires. If no refresh request is sent or received before the session expires, or if the refresh request is rejected, both parties should send BYE. The session timer feature has two new headers, Session-Expires and Min-SE, and a new response code of 422. The Session-Expires header conveys the lifetime of the session, the Min-SE header conveys the minimum value for the session timer, and the 422 response code indicates that the session timer duration is too small. If the Min-SE header is missing, the minimum value for the session timer is 90 seconds by default, in compliance with RFC 4028.

Note: The session timer applies to both 1PCC and 3PCC mode.

Session Timer Virtual Board Configuration

There are three new fields for the IP_VIRTBOARD data structure, which can be adjusted for each virtual board:

E_SIP_SessionTimer_Enabled

Enables session timer. Default is disabled. Each board has to have IP_VIRTBOARD enabled for it.

SIP_SessionTimer_SessionExpires
Sets the session expires value. Used in timer negotiation for outgoing and incoming calls. Default timer value is 1800 seconds for all calls enabled.

SIP_SessionTimer_MinSE
Sets the minimum session timer value. Used in timer negotiation for outgoing and incoming calls. Default timer value is 90 seconds for all calls enabled. A 422 response code indicates that the session timer duration is too small. That response contains a Min-SE header field identifying the minimum session interval it is willing to support.

Note: If the session timer is not enabled on the virtual board, the UPDATE method is not supported. Incoming UPDATE message is responded with a *501 Not Implemented* message.

Session Timer Negotiation

Session timer negotiation between UAS and UAC follows RFC 4028. The following SIP IP parameters have been added via the GC API for this feature. All parameters can be set on the board device, line device, or call reference number (CRN).

IPPARM_SESSION_EXPIRES
The parameter value overwrites the configured SIP_SessionTimer_SessionExpires value on virtual board. When the application makes a new call or answers an incoming call, Global Call uses this value to negotiate session interval when functioning as either a UAC or UAS. **IPPARM_SESSION_EXPIRES** sets timer value in seconds.

IPPARM_MIN_SE
The parameter value overwrites the configured SIP_SessionTimer_MinSE value on the virtual board. When the application makes a new call or answers an incoming call, Global Call uses this value to negotiate session interval when functioning as either a UAC or UAS. **IPPARM_MIN_SE** sets timer value in seconds.

Note: SIP_SessionTimer_MinSE in the virtual board is always used automatically to reject INVITE whose Session-Expires value is too small. The automatic rejection with 422 is not affected by this parameter because once the value is set in **gc_Start( )**, you can not use IPPARM_MIN_SE to change it for incoming calls. If rejected by 422 response, the application receives GCEV_DISCONNECTED with cause IPEC_SIPReasonStatus422SessionIntervalTooSmall.

IPPARM_REFRESHER_PREFERENCE
The possible values are as follows:

- IP_REFRESHER_LOCAL
- IP_REFRESHER_REMOTE
- IP_REFRESHER_DONT_CARE (default)

If set to local or remote refresher preference, the refresher parameter is added in the Session-Expires header. If set to don't care, the refresher parameter is not added in

the Session-Expires header in outgoing INVITE, and default refresher is selected, according to RFC 4028.

The actual refresher is decided by UAS and appears on 200 OK. The following table lists the refresher results if both UAS and UAC are Global Call applications using this refresher preference parameter.

UAC preference	UAS preference	Refresher
IP_REFRESHER_DONT_CARE	IP_REFRESHER_DONT_CARE	UAS
IP_REFRESHER_REMOTE	IP_REFRESHER_REMOTE	UAS
IP_REFRESHER_LOCAL	IP_REFRESHER_LOCAL	UAC
IP_REFRESHER_LOCAL	IP_REFRESHER_REMOTE	UAC
IP_REFRESHER_REMOTE	IP_REFRESHER_LOCAL	UAS

IPPARM_REFRESH_METHOD

The possible values are as follows:

- IP_REFRESH_REINVITE (default)
- IP_REFRESH_UPDATE

If selected as refresher, Global Call sends either a re-INVITE or UPDATE message to periodically refresh the session.

IPPARM_REFRESH_WITHOUT_REMOTE_SUPPORT

The possible values are as follows:

- IP_REFRESH_WITHOUT_REMOTE_SUPPORT_DISABLE
- IP_REFRESH_WITHOUT_REMOTE_SUPPORT_ENABLE (default)

The session timer mechanism can operate as long as one of the two User Agents in the call leg supports the timer extension. As call originator or terminator, the application may choose to execute the session timer if the remote side does not support the session timer. If enabled, Global Call operates the session timer if the remote side does not support session timer and sends the refresh. The other side sees the refreshes as repetitive re-INVITEs. If disabled, Global Call does not operate the session timer if the remote side does not support the session timer. When session timer is supported on only one side, re-INVITE is the only method supported in order to refresh the session. If UA uses UPDATE to refresh the session, it gets a *501 Not Implemented* message back. It is up to the application to decide if it wants to terminate the session or not once the session expires through the use of **IPPARM_TERMINATE_CALL_WHEN_EXPIRES**.

IPPARM_REFRESH_WITHOUT_PREFERENCE

The possible values are as follows:

- IP_REFRESH_WITHOUT_PREFERENCE_DISABLE
- IP_REFRESH_WITHOUT_PREFERENCE_ENABLE (default)

When the 2xx final response is received, but the refresher preference does not match the call refresher, the application may choose to execute the session timer. If enabled, Global Call operates the session timer mechanism and the refresher is different from the application preference. If disabled, Global Call does not operate the session timer.

IPPARM_TERMINATE_CALL_WHEN_EXPIRES

The possible values are as follows:

- IP_TERMINATE_CALL_WHEN_EXPIRES_DISABLE
- IP_TERMINATE_CALL_WHEN_EXPIRES_ENABLE (default)

When the session timer is about to expire, Global Call sends GCEV_SIP_SESSION_EXPIRES event to application. If enabled, Global Call sends BYE to terminate the call. If disabled, Global Call does not send BYE and the call stays connected.

Global Call IP Defines

New data structure definitions in the gcip_defs.h and gclib.h files are used by the **gc_SetUserInfo( )**, **gc_SetConfigData( )**, **gc_MakeCall( )**, and **gc_AnswerCall( )** functions to determine the duration of the call.

Event	Type	Description
GCEV_SIP_SESSION_EXPIRES	Global Call Event	Global Call event notifies application that SIP session is about to expire

Enumeration	Type	Description
IPEC_SIPReasonStatus422SessionIntervalTooSmall	eIP_EC_TYPE	eIP_EC_TYPE enumeration for SIP response code 422 Session Internal Too Small

Parameter ID	Data Type & Size	Description
IPSET_SIP_SESSION_TIMER	Global Call Set ID	Set ID used to configure SIP session timer
IPPARM_SESSION_EXPIRES	Global Call Parameter ID Type: int Size: sizeof(int)	Session-Expires timer value in seconds
IPPARM_MIN_SE	Global Call Parameter ID Type: int Size: sizeof(int)	Min-SE timer value in seconds. IPPPARM_MIN_SE does not affect Global Call decision to automatically reject incoming call with 422 Session Interval Too Small. Automatically reject decision is based only on SIP_SessionTimer_MinSE from virtual board parameter
IPPARM_REFRESHER_PREFERENCE	Global Call Parameter ID	Refresher preference
IP_REFRESHER_LOCAL	Parameter value	The User Agent wishes to be the refresher.
IP_REFRESHER_REMOTE	Parameter value	The User Agent wishes the remote side to be the refresher.
IP_REFRESHER_DONT_CARE	Parameter value	The User Agent does not care who is the refresher.
IPPARM_REFRESH_METHOD	Global Call Parameter ID	Method for refresh
IP_REFRESH_REINVITE	Parameter value	The refresh method is re-INVITE.
IP_REFRESH_UPDATE	Parameter value	The refresh method is UPDATE.

Parameter ID	Data Type & Size	Description
IPPARM_REFRESH_WITHOUT_REMOTE_SUPPORT	Global Call Parameter ID	When 2xx final response was received, but the server does not support the session timer. The application may choose to execute session timer. The session timer mechanism can be operated as long as one of the two User Agents in the call leg supports the extension. If the application decides to operate the session timer, that side sends the refresh. The other side sees the refreshes as repetitive re-INVITEs. The default behavior is to execute the session timer mechanism for the call.
IP_REFRESH_WITHOUT_REMOTE_SUPPORT_DISABLE	Parameter value	Not to execute session timer without remote support.
IP_REFRESH_WITHOUT_REMOTE_SUPPORT_ENABLE	Parameter value	Execute session timer without remote support.
IPPARM_REFRESH_WITHOUT_PREFERENCE	Global Call Parameter ID	When 2xx final response was received, but refresher preference did not match the call refresher. The application may choose to execute the session timer. If the application decides to operate the session timer mechanism, the refresher is different from the application preference. The default behavior is to execute the session timer mechanism for the call.
IP_REFRESH_WITHOUT_PREFERENCE_DISABLE	Parameter value	Not to execute session timer without preference.
IP_REFRESH_WITHOUT_PREFERENCE_ENABLE	Parameter value	Execute session timer without preference.
IPPARM_TERMINATE_CALL_WHEN_EXPIRES	Global Call Parameter ID	When the session time is about to expire. Application may choose to send BYE to terminate the call. The default behavior is to terminate the call.
IP_TERMINATE_CALL_WHEN_EXPIRES_DISABLE	Parameter value	Not to terminate the call when the session time is about to expire.
IP_TERMINATE_CALL_WHEN_EXPIRES_ENABLE	Parameter value	Terminate the call when the session time is about to expire.

Code Example

This example shows configuration of default Session-Expires and Min-SE timer values on entire virtual board using **gc_Start( )**.

```
#include "gclib.h"
..
..
#define BOARDS_NUM 1
..
..
/* initialize start parameters */
IPCCLIB_START_DATA cclibStartData;
memset(&cclibStartData,0,sizeof(IPCCLIB_START_DATA));
IP_VIRTBOARD virtBoards[BOARDS_NUM];
memset(virtBoards,0,sizeof(IP_VIRTBOARD)*BOARDS_NUM);

/* initialize start data */
INIT_IPCCLIB_START_DATA(&cclibStartData, BOARDS_NUM, virtBoards);

/* initialize virtual board */
INIT_IP_VIRTBOARD(&virtBoards[0]);

/* Enable session timer and configure default parameters */
virtBoard[0].E_SIP_SessionTimer_Enabled= ENUM_Enabled;
virtBoard[0].SIP_SessionTimer_SessionExpires= 3600;
virtBoard[0].SIP_SessionTimer_MinSE= 1800;
```

Update to Chapter 8, IP-Specific Function Information.

New functions, events, data elements, and SIP IP parameters are added for SIP PRACK handling.

SIP PRACK Handling Support (RFC 3262)

Feature Description

This feature provides the reliable transmit of a provisional response in the public SIP network. As request/response protocol for initiating and managing communication sessions, SIP uses provisional and final responses. Final responses are sent reliably and convey the result of request processing. Provisional responses provide information on the progress of request processing, but are not sent reliably in RFC.

The purpose of this feature is to handle PRACK in SIP using the Global Call API for IP. This involves the following tasks:

- Enable PRACK handling.
- Send PRACK-related messages out.
- Receive PRACK-related messages and extract the PRACK-related information
- Make PRACK message handling mandatory or optional for the remote endpoint in outbound calls.

Notes:1. This feature is supported in 3PCC mode only.

2. Refer to RFC 3262 for the Offer/Answer Model with PRACK messages.

New API Library Functions and Data Structures

This section contains information about changes and additions to the Global Call API Library.

New Functions

The following new functions support the PRACK handling feature:

- **gc_SipPrack( )** sends a PRACK message.
- **gc_SipPrackResponse( )** sends a PRACK response message.

Name: int gc_SipPrack(crn, parmblk, mode)

Inputs:

CRN crn	• call reference number
GC_PARM_BLKP parmblk	• pointer to an optional parameter block containing SDP content for the SIP PRACK message
unsigned long mode	• completion mode (EV_ASYNC)

Returns: 0 if successful
<0 if unsuccessful

Includes: gclib.h

Category: third-party call control

Mode: Asynchronous

Description

This SIP protocol specific function is used in third-party call control (3PCC) mode to send a PRACK message to the remote party of an outbound INVITE or re-INVITE call.

Parameter	Description
crn	specify a call reference number
parmblk	points to an optional parameter block containing SDP content for the SIP PRACK message. This is set to NULL if no SDP content is included in the outbound PRACK message.
mode	set to EV_ASYNC for asynchronous execution.

Termination Events

The termination events for this function are:

GCEV_SIP_PRACK_OK
Indicates that the PRACK event was sent successfully.

GCEV_SIP_PRACK_FAILED
Indicates that the PRACK message could not be sent because the state was invalid to call this function.

Example

Refer to the **Code Examples** section below.

Name: int gc_SipPrackResponse(crn, parmblk, mode)

Inputs:

CRN crn	• call reference number
GC_PARM_BLKP parmblk	• pointer to an optional parameter block containing SDP content for the SIP PRACK message
unsigned long mode	• completion mode (EV_ASYNC)

Returns: 0 if successful
<0 if unsuccessful

Includes: gclib.h

Category: third-party call control

Mode: Asynchronous

Description

This SIP protocol specific function is used in third-party call control (3PCC) mode to send a PRACK response message to the remote party of an outbound INVITE or re-INVITE call.

Parameter	Description
crn	specify a call reference number
parmblk	points to an optional parameter block containing SDP content for the SIP PRACK message. This is set to NULL if no SDP content is included in the outbound PRACK message.
mode	set to EV_ASYNC for asynchronous execution.

Termination Events

The termination events for this function are:

GCEV_SIP_PRACK_RESPONSE_OK
Indicates that the PRACK response was sent successfully.

GCEV_SIP_PRACK_RESPONSE_FAILED
Indicates that the PRACK response could not be sent because the state was invalid to call this function.

Example

Refer to the **Code Examples** section below.

New Events

The following new events are added to support PRACK handling:

GCEV_SIP_PRACK
Indicates arrival of a SIP PRACK message.

GCEV_SIP_PRACK_RESPONSE
Indicates arrival of a SIP PRACK response message.

New Data Elements

This new field for the IP_VIRTBOARD data structure can be adjusted for each virtual board:

E_SIP_PrackEnabled
Enables PRACK method. By default, PRACK is disabled. A value of ENUM_Enabled means that the application wants to enable the PRACK handling.

New SIP IP Parameters

The following SIP IP parameters have been added via the Global Call API for this feature.

IPPARM_SIP_PRACK_MANDATORY
This parameter is used as a parmid, along with a setid of IPSET_CALLINFO, when PRACK handling is enabled. It specifies whether the application wants to make PRACK handling mandatory or optional for the outbound SIP call. It also indicates if the remote endpoint has made the PRACK handling mandatory or optional for the SIP inbound call.

IP_ SIP_PRACK_MANDATORY_ON
When PRACK handling is enabled, this parameter is used as a value for the setid/parmid of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY to indicate that PRACK handling is mandatory.

IP_ SIP_PRACK_MANDATORY_OFF
When PRACK handling is enabled, this parameter is used as a value for the setid/parmid of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY to indicate that PRACK handling is optional.

Enable PRACK Handling

To enable a PRACK handling for SIP in 3PCC mode, the application sets the E_SIP_PrackEnabled field to ENUM_Enabled. The IP_VIRTBOARD structure is then passed to the **gc_Start( )** function. A setid/parmid of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY is specified in the GCEV_OFFERED and GCEV_ALERTING events parm block.

A setid/parmid/value of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY/IP_SIP_PRACK_MANDATORY_ON indicates that the remote endpoint has made the PRACK handling mandatory.

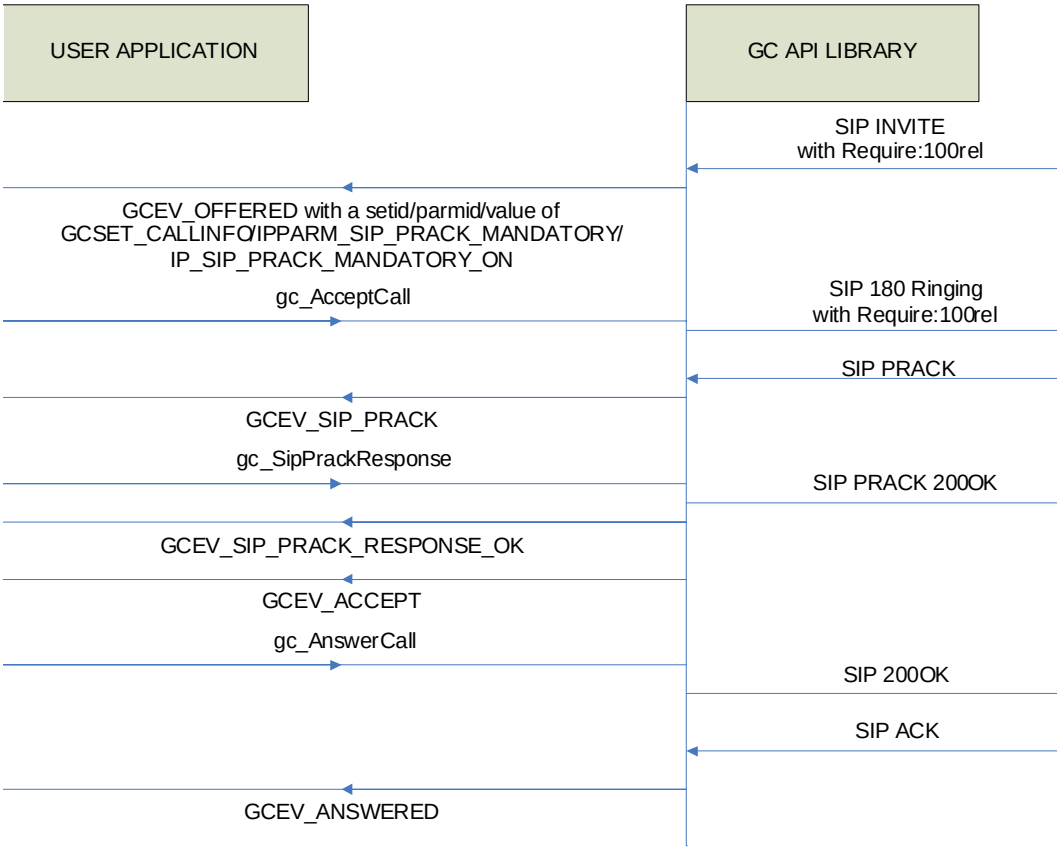
A setid/parmid/value of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY/IP_SIP_PRACK_MANDATORY_OFF indicates that the remote endpoint has made the PRACK handling optional.

When the setid/parmid combination is not present, then the remote endpoint is unable to handle PRACK messages. The call will proceed without any PRACK message exchanges.

PRACK Enabled and Remote Endpoint is Capable

If the application enables PRACK handling, and the remote endpoint is capable, the application automatically includes “Require:100rel” header in the outgoing provisional response, but excludes “100 Trying”. To send out the provisional response other than “100 Trying,” the application calls **gc_AcceptCall( )**. This causes the remote endpoint to respond with a PRACK message.

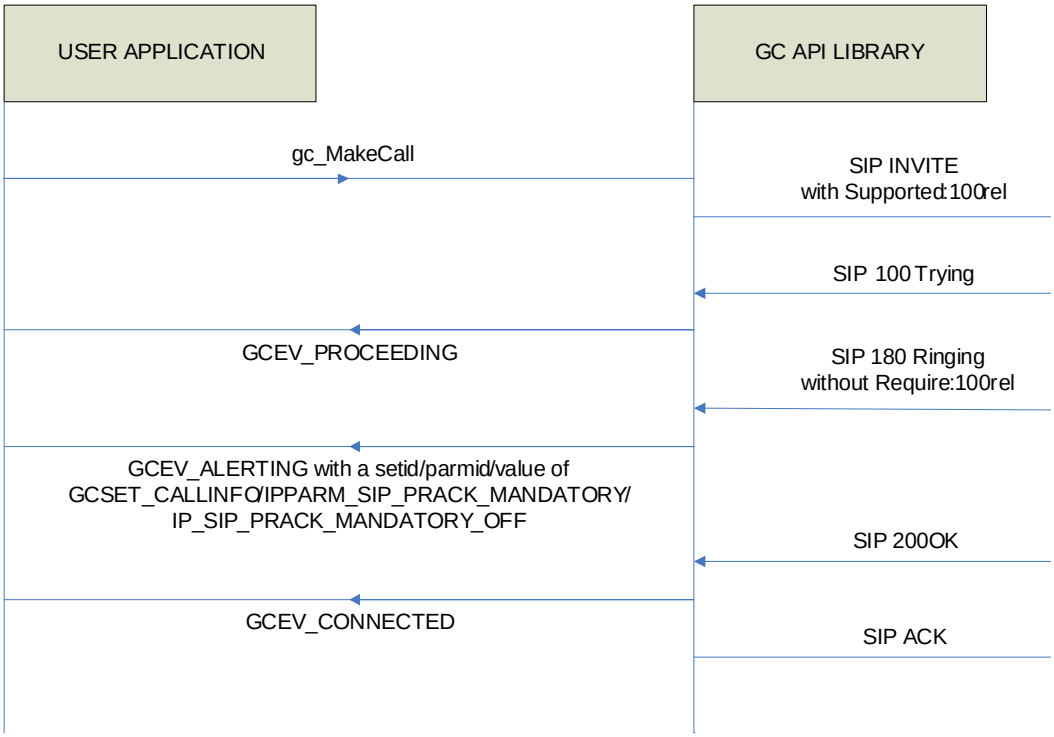
Upon receipt of a PRACK request, the application is notified with a GCEV_SIP_PRACK event. The application calls **gc_SipPrackResponse( )** to send out a PRACK response to the remote endpoint. If the PRACK response is successfully sent, a GCEV_SIP_PRACK_RESPONSE_OK is returned. A return of GCEV_SIP_PRACK_RESPONSE_FAILED indicates a failure in sending out the PRACK response.



PRACK Enabled and Remote Endpoint is Optional

If PRACK handling is enabled for outbound calls, the API will automatically include "Supported:100rel" header in the outgoing INVITE message by default making PRACK handling optional for the remote end point.

Note: The application can overwrite this default behavior on a per board basis using **gc_SetConfigData( )** or on a per call/channel basis using **gc_SetUserInfo( )** by changing the value of setid/parmid of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY. A value of IP_SIP_PRACK_MANDATORY_ON will make the PRACK handling mandatory for the remote endpoint by putting "Require:100rel" header in an outgoing INVITE message and a value of IP_SIP_PRACK_MANDATORY_OFF will make the PRACK handling optional for the remote endpoint by putting "Supported:100rel" header in an outgoing INVITE message.

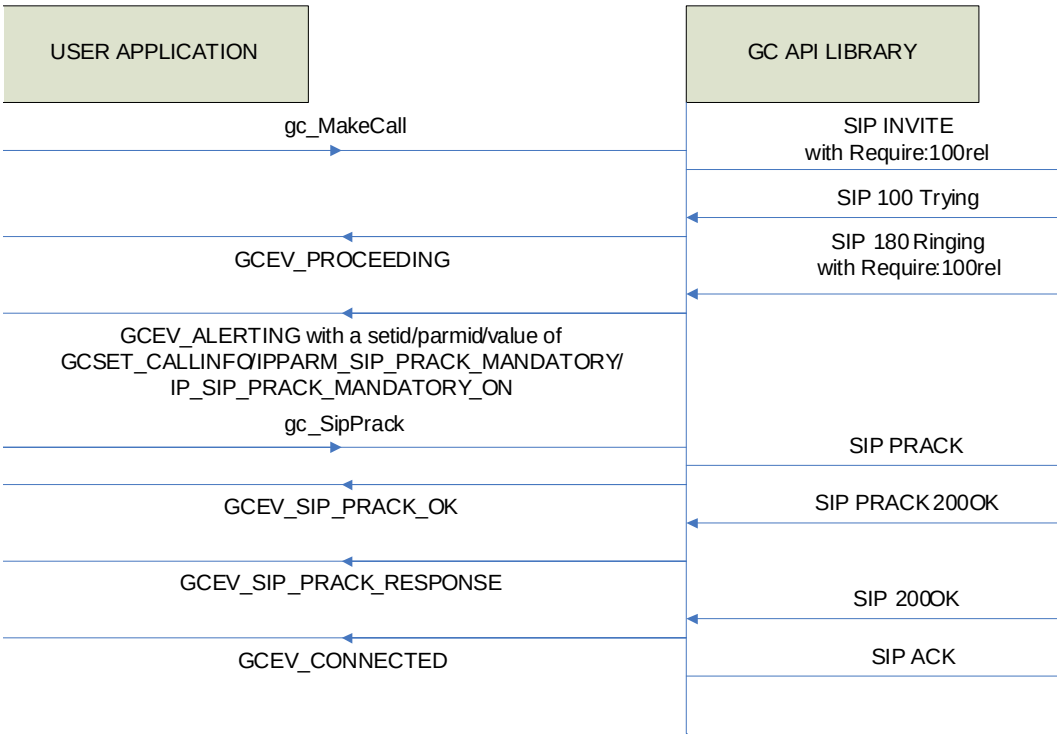


PRACK Enabled and Remote Endpoint is Mandatory

Upon receiving a provisional response other than “100 Trying”, the API will generate a GCEV_ALERTING event to the application with a setid/parmid/value of IPSET_CALLINFO/IPPARM_SIP_PRACK_MANDATORY /IP_SIP_PRACK_MANDATORY_ON if “Require:100” header is present in the incoming provisional response (excluding “100 Trying”).

The application calls **gc_SipPrack( )** to send out PRACK for this provisional response. The application is notified with either a GCEV_SIP_PRACK_OK for successfully sending out a PRACK or with a GCEV_SIP_PRACK_FAILED in case of failure in sending out a PRACK.

When the remote endpoint responds with an ACK response, the API generates a GCEV_SIP_PRACK_RESPONSE to the application.



Remote Endpoint Does Not Support PRACK

When the application sends an INVITE with “Require:100rel” to a remote site that does not support PRACK, the **gc_MakeCall( )** function fails and a GCEV_DISCONNECTED event is returned. The reason stated in the event is a bad extension.

Code Examples

Example A

The following example demonstrates how to enable PRACK handling.

```
CCLIB_START_STRUCT cclibStartStruct[] = {
{"GC_IPM_LIB", NULL},
"GC_H3R_LIB", &cclibStartData}
};
GC_START_STRUCT gcStartStruct;
IPCCLIB_START_DATA cclibStartData;
IP_VIRTBOARD virtBoards[1];

memset(&cclibStartData,0,sizeof(IPCCLIB_START_DATA));
memset(virtBoards,0,sizeof(IP_VIRTBOARD));

INIT_IPCCLIB_START_DATA(&ipcclibstart, 1, ip_virtboard);
INIT_IP_VIRTBOARD(&virtBoards[0]);

cclibStartData.num_boards = 1;
cclibStartData.board_list = virtBoards;
cclibStartData.version = 0x201;
```

```
//Set the mode as 3PCC. Prack handling can only be possible for 3PCC mode.
cclibStartData.media_operational_mode = MEDIA_OPERATIONAL_MODE_3PCC;

//Set the flag to enable the PRACK handling
virtBoards[0].E_SIP_PrackEnabled = ENUM_Enabled;

gcStartStruct.cclib_list = cclibStartStruct;
gcStartStruct.num_cclibs = GC_ALL_LIB;
gc_Start(&gcStartStruct);
```

Example B

Code to see if remote endpoint is interested in doing the PRACK handling.

```
void findoutIfRemoteIsInterestedInPRACK(METAEVENT metaevent,
GC_PARM_BLKP parm_blk)
{
  int remoteNotInterestedinPrackHandling = 1;

  switch (metaevent.evttype)
  {
       case GCEV_ALERTING:
       case GCEV_OFFERED:
         {
             GC_PARM_DATA_EXT parm;
             int rc;

             INIT_GC_PARM_DATA_EXT(&parm);

             rc = gc_util_next_parm_ex(parm_blk,&parm);
             if (rc != 0)
             {
                 printf("Remote endpoint is not interested in PRACK handling.\n");
                 return;
             }

             while (rc != EGC_NO_MORE_PARMS)
             {
               switch (parm.set_ID)
               {
                   case IPSET_CALLINFO:
                     if(parm.parm_ID == IPPARM_SIP_PRACK_MANDATORY)
                     {
                          if(*parm.pData == SIP_PRACK_MANDATORY_ON)
                          {
                             printf("Remote endpoint has made the PRACK handling mandatory.\n");
                             remoteNotInterestedinPrackHandling = 0;
                          }
                          else
                          if(*parm.pData==SIP_PRACK_MANDATORY_OFF)
                          {
                             printf("Remote endpoint has made the PRACK handling optional.\n");
                             remoteNotInterestedinPrackHandling = 0;
                          }
                     }
                     break;

                   default:
                     break;
             }
             rc = gc_util_next_parm_ex(parm_blk,&parm);
            }
        }
        break;
```

```
      default:
        return;
        break;
    }

    if(remoteNotInterestedinPrackHandling == 1)
    {
         printf("Remote endpoint is not interested in PRACK handling.\n");
    }
    return;
```

Example C

Code to make PRACK handling mandatory or optional to the remote endpoint for outbound SIP.

```
int setPrackForOutboundCall(int mode /* 1 = mandatory, 0 = optional */)
{
   GC_PARM_BLK *parmblkp = NULL;

   if(mode == 1)
   {

        /* Mandatory PRACK mode */
        gc_util_insert_parm_val(&parmblkp, IPSET_CALLINFO,IPPARM_SIP_PRACK_MANDATORY,
                                sizeof(char), IP_SIP_PRACK_MANDATORY_ON);
    }
    else
    {

        /* Optional PRACK mode */
        gc_util_insert_parm_val(&parmblkp, IPSET_CALLINFO,IPPARM_SIP_PRACK_MANDATORY,
                                sizeof(char), IP_SIP_PRACK_MANDATORY_OFF);
    }

    if(gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev, parmblkp, GC_ALLCALLS) != GC_SUCCESS)
    {
       printf("gc_SetUserInfo(linedev=%ld) Failed configuring PRACK mode for outbound call",
               port[index].ldev);
            return(-1)
    }

    gc_util_delete_parm_blk(parmblkp);

    return(0);
}
```

3.4.15 Dialogic® Global Call ISDN Technology Guide

There are currently no updates to this document.

Note: A new version (05-2242-008) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.16 Dialogic® Global Call SS7 Technology Guide

There are currently no updates to this document.

Note: A new version (05-2274-006) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.17 Dialogic® IP Media Library API Programming Guide and Library Reference

Update to the unFaultThreshold field on the IPM_QOS_THRESHOLD_DATA data structure reference page (IPY00090740)

The range for QOSTYPE_RTPTIMEOUT should be 10 to 1200 (x100 ms) instead of 50 to 1200 (x100 ms).

Update to Table 6. Quality of Service (QoS) Parameter Defaults (IPY00080727)

The following parameters shoud be modified in the table:
QoS Type Jitter, % Success Threshold 60, % Fail Threshold 40
QoS Type Lost Packets, % Success Threshold 60

3.4.18 Dialogic® Multimedia API Programming Guide and Library Reference

There are currently no updates to this document.

Note: A new version (05-2454-008) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.4.19 Dialogic® Standard Runtime Library API Library Reference

Update to **sr_getfdcnt( )** and **sr_getfdinfo( )** functions (IPY00045054).

The following caution should be added for **sr_getfdcnt( )** and **sr_getfdinfo( )** functions:

- The application must call **sr_getfdcnt**( ) and **sr_getfdinfo**( ) before calling any other Dialogic® API, if the application wants to use SELECT to retrieve SRL events on Linux.

3.4.20 Dialogic® Standard Runtime Library API Programming Guide

In Service Update 246, update to Section 2.6, “Performance Considerations”

The following paragraphs should be added to the end of this section:
“On Linux systems, creation of a new process linked to Dialogic libraries using the fork( ) or vfork( ) functions may have adverse effects with Dialogic software. The fork( ) function creates a child process and duplicates the parent's page table. The vfork( ) function creates a new process without copying the page table of the parent process; in both cases the parent is suspended until the child process is started or one of the exec( ) family of functions is invoked. The heavy usage of process-based programs can cause scheduling delays, performance degradation, and problems while copying memory mapped regions in the parent process. Please refer to your Linux operating system documentation for more information about these functions. Adverse effects with Dialogic software include delays in executing Dialogic APIs, and RTF logging while parent execution suspension happens.

Dialogic recommends creating a program using POSIX threads (pthreads) instead of spawning new processes. Pthread-based programs are more optimized because they

are created and managed with far less system overhead than running multiple processes. Also, communication between threads is achieved easier and more efficiently than communication between processes. Threads share the same memory space and are usually faster to start and terminate. Using pthreads is the preferred option when creating a program to interact with Dialogic host runtime libraries."

3.4.21 Dialogic® Voice API Library Reference

There are currently no updates to this document.

3.4.22 Dialogic® Voice API Programming Guide

There are currently no updates to this document.

3.5 Remote Control Interfaces Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- MSML Media Server Software User's Guide

3.5.1 MSML Media Server Software User's Guide

In Service Update 244, a new version (05-2513-005) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes

3.6 Demonstration Software Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® 3G-324M Multimedia Gateway Demo Guide
- Dialogic® Audio Conferencing API Demo Guide
- Dialogic® Continuous Speech Processing API Demo Guide
- Dialogic® Global Call API Demo Guide
- Dialogic® IP Gateway (Global Call) Demo Guide
- Dialogic® IP Media Server Demo Guide
- Dialogic® Multimedia Demo Guide

3.6.1 Dialogic® 3G-324M Multimedia Gateway Demo Guide

There are currently no updates to this document.

Note: A new version (05-2643-002) is now available on the documentation bookshelf. See the Revision History section of the document for a description of the changes.

3.6.2 Dialogic® Audio Conferencing API Demo Guide

There are currently no updates to this document.

3.6.3 Dialogic® Continuous Speech Processing API Demo Guide

There are currently no updates to this document.

3.6.4 Dialogic® Global Call API Demo Guide

There are currently no updates to this document.

3.6.5 Dialogic® IP Gateway (Global Call) Demo Guide

There are currently no updates to this document.

3.6.6 Dialogic® IP Media Server Demo Guide

There are currently no updates to this document.

3.6.7 Dialogic® Multimedia Demo Guide

There are currently no updates to this document.