Dialogic®
Making Innovation Thrive™

Dialogic® Host Media Processing Software Release 2.0WIN

Release Update

October 31, 2008

05-2469-042

Publication Date: October 31, 2008

Document Number: 05-2469-042

About This Publication

This section contains information about the following topics:

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

Purpose

This Release Update addresses issues associated with Dialogic® Host Media Processing Software 2.0WIN. In addition to summarizing issues that were known as of this release, the Release Update will continue to be updated to serve as the primary mechanism for communicating new issues that arise after the release date.

Intended Audience

This Release Update is intended for all users of the Dialogic® Host Media Processing Software 2.0WIN.

How to Use This Publication

This Release Update is organized into four 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 Dialogic® Host Media Processing (HMP) software. The lists include both known issues as well as issues that have been resolved since the last release. Also included are restrictions and limitations that apply to this release, as well as notes on compatibility.
- Documentation Updates: This section contains corrections and other changes that apply to the Dialogic® HMP software release documentation set that could not be made to the documents prior to the release. The 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 2.0WIN 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/support* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

Document Revision History

This revision history summarizes the changes made in each published version of the Release Update for Dialogic® Host Media Processing Software 2.0WIN, which is a document that is subject to updates during the lifetime of the release.

Document Rev 42, published October 31, 2008

Updates for Service Update 184.

In the Release Issues chapter:

- Added the following Resolved Defect: IPY00078606.

Document Rev 41, published September 16, 2008

Updates for Service Update 183.

In the Release Issues chapter:

- Added the following Resolved Defect: IPY00044978.

Document Rev 40, published August 14, 2008

Updates for Service Update 182.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00044488, IPY00044620, IPY00044700, IPY00044782.

Document Rev 39, published July 21, 2008

Updates for Service Update 181.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00042011, IPY00042653, IPY00043261, IPY00044219, IPY00044273, IPY00044437.

Document Rev 38, published June 3, 2008

Updates for Service Update 180.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00040101, IPY00042993.

Document Rev 37, published March 26, 2008

Updates for Service Update 176.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00041047, IPY00042186, IPY00042329.

Document Rev 36, published February 28, 2008

Updates for Service Update 175.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00041405, IPY00041686, IPY00041789, IPY00041876.

Document Rev 35, published January 29, 2008

Updates for Service Update 173.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00038208, IPY00039536, IPY00039823, IPY00040029, IPY00040743, IPY00041063, IPY00041112, IPY00041254, IPY00041268, IPY00041280, IPY00041296, IPY00041570, IPY00041583, IPY00041773, IPY00041815, IPY00041847, IPY00042016.
- Added the following Known Issue: IPY00036934.

Document Rev 34, Not Published

Document Rev 33, Not Published

Document Rev 32, published October 23, 2007

Updates for Service Update 160.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037476, IPY00039964, IPY00040440, IPY00040803.

Document Rev 31, published October 16, 2007

Updates for Service Update 159.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00039410, IPY00039677, IPY00039701, IPY00039707, IPY00040031, IPY00040033, IPY00040038, IPY00040082, IPY00040364, IPY00040637.

Document Rev 30, published October 30, 2007

Updates for Service Update 158.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037336, IPY00039315, IPY00039333, IPY00039401, IPY00039409, IPY00039451, IPY00039505, IPY00039564, IPY00039639, IPY00039847, IPY00039855, IPY00039965.

Document Rev 29, published September 5, 2007

Updates for Service Update 156.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00039179, IPY00039249, IPY00039649, IPY00039675, IPY00039852.

Document Rev 28, published July 30, 2007

Updates for Service Update 150.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00039036, IPY00039206, IPY00039225, IPY00039321.

In the Documentation Updates chapter:

- Updated information about the IPM_PARM_INFO data structure in Chapter 4, “Data Structures” of the Dialogic® IP Media Library API for HMP Library Reference.

Document Rev 27, published July 17, 2007

Updates for Service Update 148.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00038849, IPY00038992, IPY00039033.

Document Rev 26, published June 26, 2007

Updates for Service Update 146.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037654, IPY00038663, IPY00038708, IPY00038732, IPY00038827, IPY00038848, IPY00038856, IPY00038868, IPY00038908.

Document Rev 25, published June 18, 2007

Updates for Service Update 145.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00038218, IPY00038288, IPY00038474, IPY00038572, IPY00038580.

Document Rev 24, published June 5, 2007

Updates for Service Update 144.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00038313, IPY00038475, IPY00038513, IPY00038549.

Document Rev 23, published May 30, 2007

Updates for Service Update 143.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037358, IPY00037603, IPY00038217, IPY00038342, IPY00038343, IPY00038365, IPY00038470.

In the Documentation Updates chapter:

- Updated several sections in the Dialogic Host Media Processing Software Release 2.0WIN Software Installation Guide.

Document Rev 22, published May 8, 2007

Updates for Service Update 140.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00033127, IPY00037756, IPY00038060, IPY00038150, IPY00038240.

Document Rev 21, published April 24, 2007

Updates for Service Update 139.

In the Release Issues chapter:

Added the following Resolved Defects: IPY00036779, IPY00037699, IPY00037776.

Document Rev 20, published April 11, 2007

Updates for Service Update 137.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037422, IPY00037733, IPY00037737, IPY00037777, IPY00037778, IPY00037825.

In the Documentation Updates chapter:

- Updated the Troubleshooting chapter in the Dialogic® Host Media Processing Administration Guide to address errors received when using HMP with RealSpeak 4.0 (IPY00035708).

Document Rev 19, published April 3, 2007

Updates for Service Update 136.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00036658, IPY00037530, IPY00037541, IPY00037561, IPY00037578, IPY00037613, IPY00037633, IPY00037655.

Document Rev 18, published March 13, 2007

Updates for Service Update 134.

In the Release Issues chapter:

- Added the following Resolved Defect: IPY00034857.

Document Rev 17, published March 8, 2007

Updates for Service Update 133.

In the Post Release Developments chapter:

- Added Access ALERTING Progress Indicator Information Element

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00037172, IPY00037333.

In the Documentation Updates chapter:

- Added updates to the *Global Call IP Host Media Processing Technology Guide* for the Access ALERTING Progress Indicator Information Element.

Document Rev 16, published February 26, 2007

Updates for Service Update 131.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00034365, IPY00036223, IPY00036618, IPY00036950.

Document Rev 15, published February 21, 2007

Updates for Service Update 129.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00036021, IPY00032866. Also, added IPY00035875 (resolved in Service Update 123).

Document Rev 14, published January 31, 2007

Updates for Service Update 127.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00034254, IPY00035675, IPY00036291, IPY00036301, IPY00036346.

Document Rev 13, published January 9, 2007

Updates for Service Update 123.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00033723, IPY00033734.

Document Rev 12, published December 18, 2006

Updates for Service Update 122.

In the Post Release Developments chapter:

- Added New Parameter Enables RFC 2833 Reception Event.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00033573, IPY00034525, IPY00035631, IPY00035822.

- Also added the following Known Defect: IPY00035875.

In the Documentation Updates chapter:

- Added updates to Dialogic® Native Configuration Manager API for Windows Operating Systems Library Reference.
- Added updates to Dialogic® IP Media Library API for HMP Library Reference for RFC 2833 reception event.

Document Rev 11, published December 5, 2006

Updates for Service Update 120.

In the Post Release Developments chapter:

- Added Support for 360 Full Duplex Channels.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00034416, IPY00034840, IPY00035622, IPY00035644

In the Documentation Updates chapter:

- Added update to Release Guide about supported channels.

Document Rev 10, published November 20, 2006

Updates for Service Update 119.

In the Post Release Developments chapter:

- Added Support for 360 Full Duplex Channels.

In the Release Issues chapter:

- Added the following Resolved Defects: IPY00034309, IPY00034311,IPY00034787, IPY00035028.

Document Rev 09, published September 26, 2006

Updates for Service Update 116.

In the Release Issues chapter:

- Added the following Resolved PTRs: IPY00034499, IPY00034440, IPY00034403, IPY00034318, IPY00034316, IPY00034314, IPY00033740, IPY00033598, IPY00033552 .

In the Post Release Developments chapter:

- Added information about support for a fax gateway.

In the Documentation Updates chapter:

- Added an update to Section 4.2, “Configuration”, in the Dialogic Host Media Processing Software Release 2.0 for Windows Release Guide.
- Added an update to the **fx_initstat( )** function, describing a new Fax Gateway to the PSTN in the FAX Software Reference.
- Added information about a new Fax Gateway to Sections 3.4 and 4.24.1of the Global Call IP for HMP Technology Guide

Updates for Service Update 112.

In the Release Issues chapter:

- Added the following Resolved PTRs: IPY00033403, IPY00033474, IPY00033495, IPY00033630, IPY00033973, IPY00034196

In the Post Release Developments chapter:

- Added information about Runtime Support for Multiple PAMD/PVD Qualification Templates.
- Added information about support for Advanced Network Services.
- Added information about enhancements to Runtime Trace Facility (RTF) logging.

In the Documentation Updates chapter:

- Added a new Caution to the description of the **dx_listenEx( )** and **dx_unlistenEx( )** functions in the Voice API for Host Media Processing Library Reference .
- Added an update to Section 7.4, “Stopping HMP”, in the Dialogic Host Media Processing Software Release 2.0WIN Administration Guide.
- Added information about “Enabling Application Access to re-INVITE Requests” to section 4.7 of the Global Call IP for HMP Technology Guide

Document Rev 06, published June 26, 2006

Updates for Service Update 106.

In the Post Release Developments chapter:

- Added information about a new function, **ipm_GetCapabilities( )**.
- Removed references to Positive Voice Detection (PVD) and Positive Answering Machine Detection (PAMD) from the Feature Description section.
- Removed the fourth Use Case from Section 1.3.1, “Feature Description”.

In the Documentation Updates chapter:

- Release Guide - Added change to Table 6.
- Digital Network Interface Boards Configuration Guide - Removed previous documentation updates as an updated version (05-2474-002) has been placed on the Telecom Support Resources website at: *http://www.dialogic.com/manuals/hmp20win/default.htm*

The new version provides information about Echo Cancellation.
Also, added update to Section 6.8, “Trunk Configuration Property Sheet.”

- Diagnostics Guide - Added changes to Section 5.2 and 5.3.
- NCM API Library Reference - Added changes to the **NCM_ApplyTrunkConfiguration( )** function.
- Global Call API for HMP Programming Guide - Added changes to Section 7.2.4.

Document Rev 05, published May 22, 2006

Updates for Service Update 104.

In the Release Issues chapter:

- Added the following Resolved PTR: IPY00032398

In the Post Release Developments chapter:

- Added information about dynamically retrieving and modifying selected protocol parameters.

In the Documentation Updates chapter:

- Updated the Global Call CDP Configuration Guide, Section 2.4.2, Downloading the Protocol and CDP File on a Windows System.
- Updated the following chapters in the Audio Conferencing API Programming Guide: Application Development Guidelines, Active Talker, and Background Music.
- Updated the **dcb_getbrdparm( )** and **dcb_setbrdparm( )** functions in the Audio Conferencing API Library Reference.
- Updated the Device Management API for Windows and Linux Operating Systems Library Reference. The Error Codes chapter and applicable functions were updated to ignore references to error codes that have not yet been implemented.

Document Rev 04, published May 5, 2006

Updates for Service Update 99.

In the Post Release Developments chapter:

- Added information about the support for MIME on Subscribe and Notify messages.

In the Documentation Updates chapter:

- Added a note about multiple NIC cards to Chapter 2 of the Administration Guide.
- Added a subsection to Chapter 11 of the Administration Guide describing how to troubleshoot a failure to start in a boardless system.
- Added information about the **dcb_estconf( )** and **dcb_addtoconf( )** functions in the Audio Conferencing API Library Reference.

Document Rev 03, published April 17, 2006

Update for Service Update 97.

In the Release Issues chapter:

- This section has been modified to show issues by Change Control System defect number and by PTR number. Issues reported prior to March 27, 2006, will be identified by both numbers. Issues reported after March 27, 2006, will only have a defect number.
- Added the following Resolved PTRs: IPY00032239 (PTR 36769)

In the Documentation Updates chapter, updated the following documents:

- Global Call ISDN Technology Guide - Updated Sections 8.3.17 and 10.

Document Rev 02, published March 9, 2006

Update for Service Update 94.

Added a new chapter, Post Release Developments, that describes new features contained in the Service Update. These new features include:

- Support for V.17 PCM Fax
- Mixed ISDN, CAS, and R2MF Protocols

In the Release Issues chapter:

- Added the following Resolved PTRs: 36730, 36748

In the Documentation Updates chapter, updated the following documents:

- Release Guide - Updated Section 3.1
- Digital Network Interface Boards Configuration Guide - Updated Section 5.8
- Administration Guide - Updated Sections 2.1, 2.4, 3.1, 3.4, 4.1 and Chapter 11
- Global Call ISDN Technology Guide - Updated Sections 4.12, 8.2.18.2, and 8.2.30.
- IP Media Library API for HMP Library Reference - Revised definition of NLP in Table 2 for the PARMCH_ECNLP_ACTIVE parameter, and revised the IPMEV_QOS_ALARM event description.

Document Rev 01, published December 2005

Initial Version of document.

Post Release Developments 1

This section describes significant changes to the Dialogic® Host Media Processing Software 2.0WIN subsequent to the general availability release.

1.1 Service Update for Dialogic® HMP Software Release 2.0WIN

This Service Update for Dialogic® Host Media Processing Software 2.0WIN 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.

For information about installing this Service Update, refer to the *Dialogic® Host Media Processing Software Release 2.0 WIN Installation Guide*.

1.2 Access ALERTING Progress Indicator Information Element

With this Service Update, the application has the ability to set and/or retrieve the Progress Indicator Information Element (PIIE) in an H.323 ALERTING message. The application can also disable the dispatch of a PROGRESS message after an ALERTING message is received.

1.2.1 Feature Description

This feature allows the application to populate and send a new PIIE and include it in the ALERTING response to an incoming call SETUP message. Currently, the Call Control Library automatically sends a PROGRESS message after every ALERTING message. With this feature, a new parameter is defined to optionally disable an automatic PROGRESS message in the virtual board. Once disabled, PROGRESS will not be sent after receipt of an ALERTING message. By default, this feature is disabled and current behavior is preserved.

The Global Call functions **gc_SetUserInfo( )** and **gc_GetMetaEvent( )** are used to set and get PIIE in an ALERTING message. Both functions use the GC_PARM_BLK and GC_PARM_DATA data structures.

The following new parameter ID, IPPARM_H323_AUTO_PROGRESS_DISABLE, is added to disable an H.323 automatic PROGRESS message after ALERTING on a virtual board.

Parameter ID	Data Type & Size	Description	SIP/ H.323
IPPARM_H323_AUTO_PROGRESS_DISABLE	Type: None Size: 0	Parameter ID in IPSET_CONFIG. This parameter is used to disable H323 automatic PROGRESS message after ALERTING on virtual board.	H.323

1.2.2 Enable Access to H.323 IE

In order to access H.323 IE, the IP_H323_MSGINFO_ENABLE must be enabled in h323_msginfo_mask on a virtual board at **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]);

// IP_H323_MSGINFO_ENABLE must be set to activate this feature
virtBoard[bid].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE
```

1.2.3 Disable an Automatic PROGRESS Message

The following code illustrates how to disable an automatic PROGRESS message on a virtual board:

```
#include "gclib.h"
..
..
GC_PARM_BLK *pParmBlock = NULL;
long t = 0;

gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_CONFIG,
                        IPPARM_H323_AUTO_PROGRESS_DISABLE,
                        sizeof(int),
                         0);

// Set config data
gc_SetConfigData(GCTGT_CCLIB_NETIF,
                 boarddev,
                 pParmBlock,
                 0,
                 GCUPDATE_IMMEDIATE,
                 &t,
                 EV_ASYNC);

gc_util_delete_parm_blk(pParmBlock);
```

1.2.4 Setting the PIIE

PIIE is set using the Global Call API **gc_SetUserInfo( )** function, but information is not transmitted until calling **gc_AcceptCall( )**. The application will overwrite the PIIE if any incoming ALERTING or PROGRESS messages are received for the same channel.

Code Example

```
#include "gclib.h"
..
..
GC_PARM_BLK *pParmBlock = NULL;
unsigned char progress_ind[] = {0x1E,0x02,0x80,0x81};

//set up progress indicator
gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_CALLINFO,
                        IPPARM_PROGRESS_IND,
                        sizeof(progress_ind),
                        progress_ind);


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

gc_util_delete_parm_blk(pParmBlock);
gc_AcceptCall(crn, NULL, EV_ASYNC);
```

1.2.5 Retrieving PIIE

The PIIE values are reported to the application through events processed using the **gc_GetMetaEvent( )**. The following example illustrates how to retrieve PIIE from the GCEV_ALERTING event after receiving an ALERTING message.

Code Example

```
#include "gclib.h"
..
..
METAEVENT  metaevt;
GC_PARM_BLK *pParmBlock = NULL;
GC_PARM_DATA  *parmp = NULL;

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

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

               /* going thru each parameter block data*/
               while ((parmp = gc_util_next_parm(pParmBlock,parmp)) != 0)
               {
                       switch (parmp->set_ID)
                       {
                       /* Handle the extended information */
                       case IPSET_CALLINFO:
                               switch (parmp->parm_ID)
                               {
                               case IPPARM_PROGRESS_IND:
                               if(parmp->data_size != 0)
                               {

                                      printf("\tGot PIIE: ");
                                      for(unsigned int pii= 0;pii<parmp->data_size;pii++)
                                      printf ("0x%x ",*(((unsigned char*)(parmp->pData))+pii));
                                      printf ("\n");
                                }
                                }
                                 break;
                         }
                         break;
                   }
          .
          .
          .
}
```

1.2.6 Documentation

The online bookshelf provided with Dialogic® Host Media Processing Software 2.0WIN contains information about all release features including features for application development, configuration, administration, and diagnostics.

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

- *Global Call IP for Host Media Processing Technology Guide*
- *Global Call API for Host Media Processing Programming Guide*
- *Global Call API Library Reference*

1.3 New Parameter Enables RFC 2833 Reception Event

With the Service Update, there is now a preferred alternative to enable RFC 2833 information over the IP network in HMP systems using an existing function with a new parameter. The preferred new parameter has the same functionality as the current parameter in that it enables RFC 2833 information reception and receives the information in the form of an unsolicited event from the HMP software into a user application. The parameter is also associated with a new event and data structure. The current parameter, event, and data structure are being deprecated.

1.3.1 Feature Description

The preferred method for enabling RFC 2833 DTMF information is to use the **ipm_EnableEvents( )** function with a new enumeration value for pEvents, EVT_TELEPHONY, instead of the current value, **EVT_RFC 2833**.

When RFC 2833 information is received by the HMP software over the IP network, a new event is generated, IPMEV_TELEPHONY_EVENT, instead of the current event IPMEV_RFC2833SIGNALRECEIVED. A new data structure, IPM_TELEPHONY_INFO, is also associated with the IPMEV_TELEPHONY_EVENT event instead of the current data structure, IPM_RFC2833_SIGNALID_INFO. The IPM_TELEPHONY_EVENT_INFO and the IPM_TELEPHONY_TONE_INFO are members of the IPM_TELEPHONY_INFO.

Note: Tone signal (IPM_TELEPHONY_TONE_INFO) information is not currently suppported; only IPM_TELEPHONY_EVENT_INFO is supported.

IPM_TELEPHONY_INFO

```
typedef struct ipm_telephony_info_tag
{
   unsigned long              UnVersion;      /* Structure version for library use only */
   eIPM_TELEPHONY_INFO_TYPE    eTelInfoType;   /* RFC2833 Info type - named event or tone */
   union
   {
      IPM_TELEPHONY_EVENT_INFO  TelEvtInfo;   /* RFC2833 named event info eg. DTMF digit * /
      IPM_TELEPHONY_TONE_INFO   TelToneInfo;  /* RFC2833 non-standard tone signal Information */
   }TelephonyInfo;
} IPM_TELEPHONY_INFO, *PIPM_TELEPHONY_INFO;
```

Description

This structure contains telephony information (such as RFC 2833 information) that is to be transferred over an IP network.

Field Descriptions

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

UnVersion
: version of the IPM_TELEPHONY_INFO structure. This field is used by the IP Media Library for checking the backward binary compatibility of future versions of the data structure.

eTelInfoType
: the information type, for example, an RFC 2833 named event or tone

  The eIPM_TELEPHONY_INFO_TYPE data type is an enumeration which defines the following values:
  - TEL_INFOTYPE_EVENT – indicates that the union in this structure is the IPM_TELEPHONY_EVENT_INFO structure.
  - TEL_INFOTYPE_TONE – indicates that the union in this structure is the IPM_TELEPHONY_TONE_INFO structure. Reserved for future use.

TelephonyInfo.TelToneInfo
: non-standard tone signal information, for example, an RFC 2833 non-standard tone. Reserved for future use.

TelephonyInfo.TelEvtInfo
: named event information, for example, RFC 2833 DTMF digit. See IPM_TELEPHONY_EVENT_INFO for more information.

IPM_TELEPHONY_EVENT_INFO

```
typedef struct ipm_telephony_event_info_tag
{
unsigned int             unVersion;          /* Structure version for library use only */
eIPM_TELEPHONY_EVENT_ID  eTelephonyEventID;  /* The named event usually DTMF named event */
short                    sVolume;            /* The power level for the DTMF event tone*/
unsigned short           usDuration;         /* Duration for the DTMF digit in ms*/
} IPM_TELEPHONY_EVENT_INFO, *PIPM_TELEPHONY_EVENT_INFO;
```

■ Description

The IPM_TELEPHONY_EVENT_INFO data structure contains detailed information about a telephony event, for example a DTMF event. This structure is a child structure of the IPM_TELEPHONY_INFO data structure.

The following inline function is provided to initialize the IPM_TELEPHONY_EVENT_INFO structure:

```
void INIT_IPM_TELEPHONY_EVENT_INFO(PIPM_TELEPHONY_EVENT_INFO pIpmTelEvtInfo)
{
   memset((IPM_TELEPHONY_EVENT_INFO *) pIpmTelEvtInfo, 0,
       sizeof(IPM_ TELEPHONY_EVENT_INFO));
   pIpmTelEvtInfo->version = IPM_TELEPHONY_EVENT_INFO_VERSION;
}
```

The IPM_TELEPHONY_EVENT_INFO_VERSION define is used to initialize the version field in the structure. The initial value is 0x000, but the value will change if the structure is modified in the future.

■ Field Descriptions

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

UnVersion
: version of the IPM_TELEPHONY_EVENT_INFO structure. This field is used by the IP Media Library for checking the backward binary compatibility of future versions of the data structure.

eTelephonyEventID
: a named event, typically a DTMF named event. The datatype of the telephony_event field is an eIPM_TELEPHONY_EVENT_ID enumeration that lists all possible tone signal identifiers as described in RFC 2833. The eIPM_TELEPHONY_EVENT_ID is an enumeration with values listed as follows:

- SIGNAL_ID_EVENT_DTMF_0
- SIGNAL_ID_EVENT_DTMF_1
- SIGNAL_ID_EVENT_DTMF_2
- SIGNAL_ID_EVENT_DTMF_3
- SIGNAL_ID_EVENT_DTMF_4
- SIGNAL_ID_EVENT_DTMF_5
- SIGNAL_ID_EVENT_DTMF_6

- SIGNAL_ID_EVENT_DTMF_7
- SIGNAL_ID_EVENT_DTMF_8
- SIGNAL_ID_EVENT_DTMF_9
- SIGNAL_ID_EVENT_DTMF_STAR
- SIGNAL_ID_EVENT_DTMF_POUND
- SIGNAL_ID_EVENT_DTMF_A
- SIGNAL_ID_EVENT_DTMF_B
- SIGNAL_ID_EVENT_DTMF_C
- SIGNAL_ID_EVENT_DTMF_D

sVolume
: the power level associated with the DTMF event tone

usDuration
: the duration of the DTMF digit in milliseconds

Code Example

```
#include <stdio.h>
#include <string.h>
#include <srllib.h>
#include <ipmlib.h>

void main()
{
   int nDeviceHandle;
   eIPM_EVENT myEvents[2] ={EVT_TELEPHONY, EVT_JITTER};
   // Register event handler function with srl
   sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
   /*  Main Processing
    */
   /* Need to enable two events for IP device handle, nDeviceHandle.
       ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
   */
   if(ipm_EnableEvents(nDeviceHandle, myEvents, 2, EV_ASYNC) == -1)
   {
      printf("ipm_EnableEvents failed for device name %s with error = %d\n",
      ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
      /*  Perform Error Processing
          ..
     */
   }
   /* Continue Processing */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void *pVoid   = sr_getevtdatap();
    IPM_TELEPHONY_INFO *pTelInfo;

    switch(nEventType)
    {
        /* List of expected events */
        /* Expected reply to ipm_EnableEvents() */
     case IPMEV_EVENT_ENABLED:
           printf("Received IPMEV_EVENT_ENABLED for device = %s\n",  ATDV_NAMEP(nDeviceID));
       break;

        /* Received unsolicited Telephony event (RFC2833 info). */
     case IPMEV_TELEPHONY_EVENT:
            printf("Received IPMEV_TELEPHONY_EVENT for device name = %s\n",
ATDV_NAMEP(nDeviceID));
            pTelInfo = (IPM_TELEPHONY_INFO*)pVoid;

            switch (pTelInfo->eTelInfoType)
            {
            case TEL_INFOTYPE_EVENT:
                  printf("Telephony Info Type = RFC2833 NAMED EVENT INFO!!\n");
                  printf("Telephony Named Event ID = %d\n", pTelInfo-
>TelEvtInfo.eTelephonyEventID);
                  printf("Named Event Volume = %d dB\n", pTelInfo->TelEvtInfo.sVolume);
                  printf("Named Event Duration = %d ms\n", pTelInfo->TelEvtInfo.usDuration);
              break;
            case TEL_INFOTYPE_TONE:
                  printf("Telephony info type TEL_INFOTYPE_TONE is not supported.\n");
              break;
            default:
                  printf("Unknown telephony info type.\n");
              break;
            }
            break;

        default:
                printf("Received unknown event = %d for device = %s\n", nEventType,
ATDV_NAMEP(nDeviceID));
            break;
        }
}
```

1.3.2 Documentation

The online bookshelf provided with Dialogic® Host Media Processing Software 2.0WIN contains information about all release features including features for application development, configuration, administration, and diagnostics.

For more information about receiving RFC 2833 DTMF information, see the following document:

- *IP Media Library API for Host Media Processing Library Reference*

1.4 Support for 360 Full Duplex Channels

With the Service Update, the digital network interface boards running on Dialogic® Host Media Processing Software Release 2.0WIN support a maximum of 360 full duplex channels per system. The previous configuration was 240 channels.

1.5 Global Call API Access to New H.323/Q.931 Message IEs

With the Service Update, you can access and configure the called and calling party number information elements (IEs) and various subfields of the H.322/Q.931 SETUP message.

1.5.1 Feature Description

You now have the ability to access additional fields in the calling party number (CGPN) and called party number (CDPN) IEs within a H.225/Q.931 SETUP message when using H.323 IP call signaling.

The SETUP message is a standard call signaling message used by a calling H.323 entity to establish a connection with the called entity. This message is currently supported with limited access to information fields in the CGPN and CDPN IEs. Presently only the Called/Calling Party number can be modified by the application via **gc_SetUserInfo( )** or when invoking the **gc_MakeCall( )** function. You now can use an existing parameter set ID and its new parameter IDs to send and receive these CPN fields via Global Call over an IP network.

1.5.2 New Parameter IDs

An existing parameter set ID (IPSET_CALLINFO) and new parameter IDs support the CPN information as shown in the table below:

IPSET_CALLINFO Parameter IDs

Parameter ID	Set	Sent	Retrieve
IPPARM_CGPN_TYPE_OF_NUMBER IPPARM_CDPN_TYPE_OF_NUMBER IPPARM_CGPN_NUMBERING_PLAN_ID IPPARM_CDPN_NUMBERING_PLAN_ID IPPARM_CGPN_SCREENING_INDICATOR IPPARM_CGPN_PRESENTATION_INDICATOR	GC_PARM_BLK **gc_SetUserInfo( )**	**gc_MakeCall( )**	**gc_Extension( )** (IPEXTID_GETINFO) and asynchronous GCEV_EXTENTIONCMPLT completion event

Parameter ID Details

Parameter ID	Data Type & Size	Description
IPPARM_CGPN_TYPE_OF_NUMBER IPPARM_CDPN_TYPE_OF_NUMBER	Type: unsigned char Size: 1 byte	Contains the type of number in the CGPN or CDPN
IPPARM_CGPN_NUMBERING_PLAN_ID IPPARM_CDPN_NUMBERING_PLAN_ID	Type: unsigned char Size: 1 byte	Contains the numbering plan identification in the CGPN or CDPN
IPPARM_CGPN_SCREENING_INDICATOR	Type: unsigned char Size: 1 byte	Contains the screening indicator in the CGPN
IPPARM_CGPN_PRESENTATION_INDICATOR	Type: unsigned char Size: 1 byte	Contains the presentation indicator in the CGPN

The following definitions are in the *gcip_defs.h* file:

```
#define IPPARM_ CGPN_TYPE_OF_NUMBER           0x13
#define IPPARM_ CDPN_TYPE_OF_NUMBER           0x14
#define IPPARM_ CGPN_NUMBERING_PLAN_ID        0x15
#define IPPARM_ CDPN_NUMBERING_PLAN_ID        0x16
#define IPPARM_ CGPN_SCREENING_INDICATOR      0x17
#define IPPARM_ CGPN_PRESENTATION_INDICATOR   0x18
```

The following data variables are in the *gcip.h* file:

```
typedef unsigned char   CPN_TON;    /* Type of number */
typedef unsigned char   CPN_NPI;    /* Numbering plan identification */
typedef unsigned char   CPN_SI;     /* Screening Indicator      */
typedef unsigned char   CPN_PI;     /* Presentation Indicator */
```

1.5.3 Enabling the Setting and Retrieving of Q.931 Message IEs

To enable the setting and retrieving of all supported Q.931 message IEs, set the h323_msginfo_mask field in the IP_VIRTBOARD structure to a value of IP_H323_MSGINFO_ENABLE for each IPT board device **before** calling **gc_Start( ).**

Note: By default, the underlying H.323 stack is not enabled to receive incoming Q.931 message IEs.

A code snippet showing how to do this is given below:

```
IP_VIRTBOARD virtBoard[MAX_BOARDS];
memset(virtBoard,0,sizeof(IP_VIRTBOARD) * MAX_BOARDS);
bid = 1;
INIT_IP_VIRTBOARD(&virtBoard[bid]);
// fill up other board parameters
….
 virtBoard[bid].localIP.ip_ver = IPVER4;
 virtBoard[bid].localIP.u_ipaddr.ipv4 = (unsigned int) IP_CFG_DEFAULT;
…
virtBoard[1].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE;
//Then use the virtBoard structure in the gc_Start() function appropriately
```

You can also enable reception of other H.323 fields simultaneously via the code:

```
virtBoard[1].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE | IP_H323_ANNEXMMSG_ENABLE;
```

1.5.3.1 Stopping the Reception of CPN Information

Currently, there is no way for the user application to turn off the reception of Q.931 IEs received by the underlying H323 stack once they are enabled, without stopping the application or restarting the stack.

1.5.4 Setting Up CPN fields in the GC_PARM_BLK Data Structure

Before calling the **gc_MakeCall( )** function, you must set up the CPN fields to be included in the GC_PARM_BLK data structure. The GC_PARM_BLK should include the existing parameter set ID IPSET_CALLINFO and the newly defined parameter IDs described in the New Parameter IDs section, which specifies which CPN fields are to be set in the parameter block structure of a Make Call block.

To set up the CPN fields in the GC_PARM_BLK structure, call the **gc_util_insert_parm_ref( )** function.

Code Example

The following is an example of how to specify the CPN fields for sending.

```
#include <stdio.h>
#include <string.h>
#include <gcip.h>
#include <.h>


void main()
{
   CPN_TON    cgpn_ton, cdpn_ton;
   CPN_NPIcgpn_npi, cdpn_npi;
   CPN_SIcgpn_si;
   CPN_PIcgpn_pi;
   GC_PARM_BLKPpParmBlock;

  /*. .   Main Processing….*/
  /*  Set CPN fields in the Make Call Block to be sent out via SETUP message */

  cgpn_ton = 0x1;   // Note that the field values must be valid.
  cdpn_ton = 0x4;
  cgpn_npi = 0x1;
  cdpn_npi = 0x1;
  cgpn_si = 0x0;
  cgpn_pi = 0x0;

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CGPN_TYPE_OF_NUMBER,
                            sizeof(unsigned char),
                            &cgpn_ton);

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CDPN_TYPE_OF_NUMBER,
                            sizeof(unsigned char),
                            &cdpn_ton);

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CGPN_NUMBERING_PLAN_ID,
                            sizeof(unsigned char),
                            &cgpn_npi);

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CDPN_NUMBERING_PLAN_ID,
                            sizeof(unsigned char),
                            &cdpn_npi);

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CGPN_SCREENING_INDICATOR,
                            sizeof(unsigned char),
                            &cgpn_si);

    gc_util_insert_parm_ref(&pParmBlock,
                            IPSET_ CALLINFO,
                            IPPARM_CGPN_PRESENTATION_INDICATOR,
                            &cgpn_pi);

  /*. .. Continue Main processing. … call gc_MakeCall() */
}
```

1.5.5 Generating CPN Fields in a SETUP Message

If you choose to insert the CPN data into the GC_MAKECALL_BLK using the **gc_SetUserInfo( )** function, refer to the *Global Call IP for Host Media Processing Technology Guide* for details on how this can be done.

After setting the CPN signaling message fields as described in the Setting Up CPN fields in the GC_PARM_BLK Data Structure section, the user application calls the **gc_MakeCall( )** function, passing it a pointer to the GC_MAKECALL_BLK.

1.5.6 Retrieving CPN Information

When the underlying H.323 stack is enabled to retrieve incoming call info fields, including CPN information, any CPN fields detected in an associated H.225 message will be retrieved and stored in Global Call.

Use the **gc_Extension( )** function to retrieve the CPN data within any valid incoming H.225 message containing CPN fields, while a call is in any state, after the OFFERED state. This is similar to receiving other call related information via the GCEV_EXTENSIONCMPLT event received as a termination event to the **gc_Extension( )** function.

Set the target_type to GCTGT_GCLIB_CRN. Set the target_id to the actual CRN. If incoming CPN data is available, the information is included with the corresponding GCEV_EXTENSIONCMPLT asynchronous termination event.

The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSIONCMPLT event is a pointer to an EXTENSIONEVTBLK structure that contains a GC_PARM_BLK with the requested CPN information.

When trying to retrieve the CPN information, it is necessary to specify each of the CPN parameters in the extension request, for which information from the stack is needed.

Code Examples

Specifying CPN Field for Receiving

A code example of how to specify the CPN fields for receiving is shown below. This example is just for the Calling Number Type of Number field. The method to specify the other CPN data would be similar.

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

   frc = gc_util_insert_parm_val(&gcParmBlk,
                                 IPSET_CALLINFO,
                                 IPPARM_CGPN_TYPE_OF_NUMBER,
                                 sizeof(unsigned char),1);
   if (GC_SUCCESS != frc)
   {
       return GC_ERROR;
   }

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

   gc_util_delete_parm_blk(gcParmBlk);

    return GC_SUCCESS;

}
```

Retrieving CPN Information

A code example of how to extract CPN information from an unsolicited GCEV_EXTENSIONCMPLT event received as a result of a request for call-related information is shown below:

```
int OnExtension(GC_PARM_BLKP parm_blk,CRN crn)
{
   GC_PARM_DATA *parmp = NULL;
   parmp = gc_util_next_parm(parm_blk,parmp);

   if (!parmp)
   {
      return GC_ERROR;
   }

   while (NULL != parmp)
   {
     switch (parmp->set_ID)
     {
       case IPSET_CALLINFO:
           switch (parmp->parm_ID)
           {
             case IPPARM_CGPN_TYPE_OF_NUMBER:
                   printf("\tReceived CPN data Calling Party Type of Number: %d\n", (*(unsigned
char*)(parmp->value_buf)));
             break;

             case IPPARM_CDPN_TYPE_OF_NUMBER:
                   printf("\tReceived CPN data Called Party Type of Number: %d\n", (*(unsigned
char*)(parmp->value_buf)));
             break;

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

```
            break;
          }
   break;
     }
   parmp = gc_util_next_parm(parm_blk,parmp);
   }
}
```

1.5.7 Documentation

The online bookshelf provided with Dialogic® Host Media Processing Software 2.0WIN contains information about all release features including features for application development, configuration, administration, and diagnostics.

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

- *Global Call IP for Host Media Processing Technology Guide*
- *Global Call API for Linux and Windows Operating Systems Library Reference*

For more information about the IP Media API, see the following documents:

- *IP Media Library API for Host Media Processing Programming Guide*
- *IP Media Library API for Host Media Processing Library Reference*

1.6 Support for a FAX Gateway

The Service Update provides an enhancement to provide fax T.38/V.17 Gateway capabilities. The Fax Gateway allows Fax data to be received via one transport mechanism and then converts and sends the data via a different transport type. This will allow a host running HMP 2.0 to pass Fax data from an IP network (via T.38 protocol) to the PSTN (via V.17) and vice versa. This enhancement provides the programming mechanism to access this new feature, including the ability to start the gateway process and to be notified when the gateway process completes.

The T.38/V.17 gateway provides a means to convert one form of fax transmission to the other, such that each receiving end is unaware of the originating format. For the HMP Windows implementation, the conversion processing takes place in the HMP fax firmware layer. No image processing is done on the data, nor is there any T.30 protocol processing.

The API and event support needed to enable gateway processing include the following:

- A new defined protocol state value that is passed to the function **fx_initstat( )**. The new value will initiate a gateway session. The new value is: DF_T38GW.
- A new FAX event type that is returned to an enabled application. The new event type will provide notification when the gateway session completes. The new event type is: TFX_T38GW.
- New gateway error values that are returned from ATFX_ESTAT when problems arise. The new error values (and their meaning) are as follows:
    - EFX_SIGNALTIMEOUT - *Signal time-out - no data or events received during GW session*

- EFX_DCNTIMEOUT - *DCN timeout - GW session almost complete but no DCN received*
- EFX_BADIPADDRESS - *Bad IP address - T38 subsystem did not get remote IP address - check R4 application*
- EFX_CTBUSERROR -*CTBus error w/TDM portion of GW session - check R4 application*

1.7 Runtime Support for Multiple PAMD/PVD Qualification Templates

With the Service Update, it is possible to define up to five different positive answering machine detection (PAMD) qualification templates and five different positive voice detection (PVD) qualification templates for use at runtime, for improved call progress analysis.

1.7.1 Feature Description

For Dialogic® HMP applications that are not based on the Global Call API, this feature enables you to define a maximum of five PAMD and five PVD qualification templates for improved call progress analysis. These templates are defined in the CONFIG file and are downloaded to the firmware when the HMP software is started. One template at a time is chosen for call progress analysis through the updated **dx_dial( )** function and updated DX_CAP structure in the voice library.

Updated dx_dial( ) Function

The **dx_dial( )** function dials an ASCIIZ string on an open, idle channel and enables call progress analysis to provide information about a call. This function points to the DX_CAP structure, which has a newly supported field that contains the template ID of the PAMD or PVD qualification template to be used for a particular call.

By default, this function uses default PAMD and PVD templates stored in the firmware. The template ID for the default PAMD template is 106561 (0x1a041). The template ID for the default PVD template is 128193 (0x1f4c1).

Updated DX_CAP Structure

The DX_CAP structure now supports the ca_pamd_qtemp field. This field specifies the template ID for the PAMD or PVD qualification template. The field is one byte, but the template ID is defined as long; thus, the template ID is reduced to 1 byte long by the application before it is passed to **dx_dial( )**.

Updated CONFIG File

The signal detector [SigDet] section of the CONFIG file may include qualification templates for positive answering machine detection (PAMD) and positive voice detection (PVD) used in call progress analysis.

If the application uses the default templates, no template definitions need to be included in the CONFIG file. If the application uses user-defined templates, these templates are defined in the CONFIG file.

For more information on the [SigDet] section, see the documentation update provided in section 3.2.2, "Digital Network Interface Boards Configuration Guide," Update to Chapter 7, "CONFIG File Parameter Reference." This update also points to a tech note on the technical support web site that describes how to tune or create new user-defined templates.

The following PAMD template IDs are available:

Pre-Defined or User-Defined	PAMD Template ID (Hexadecimal)	PAMD Template ID (Decimal)
Pre-defined (default)	0x1a041	106561
User-defined	0x1a044	106564
User-defined	0x1a045	106565
User-defined	0x1a046	106566
User-defined	0x1a047	106567

The following PVD template IDs are available:

Pre-Defined or User-Defined	PVD Template ID (Hexadecimal)	PVD Template ID (Decimal)
Pre-defined (default)	0x1f4c1	128193
User-defined	0x1f4c2	128194
User-defined	0x1f4c3	128195
User-defined	0x1f4c4	128196
User-defined	0x1f4c5	128197

Example Code

The following example code snippet illustrates use of the ca_pamd_qtemp field and template ID. Note that the field is one byte, but the template ID is defined as long; thus, only the last byte of the template ID is specified.

```
[...]

DX_CAP cap_s;
int TemplateID = 0x1f4c2;
cap_s.ca_pamd_qtemp = (TemplateID & 0xFF);
```

```
if ((car = dx_dial(ddd, dialstrg, (DX_CAP *) & cap_s, DX_CALLP|EV_SYNC)) == -1)
{
     /* handle error */
}
```

1.7.2 Implementation Guidelines

The following guidelines apply when specifying a user-defined PAMD or PVD qualification template for call progress analysis using **dx_dial( )** and DX_CAP:

- If the existing default templates do not achieve the desirable success rate for detecting PAMD or PVD, you can tune the default templates or define new templates in the CONFIG file. A maximum of five templates for PAMD and five templates for PVD can be defined.

  For details on how to tune and define new templates, see the tech note at http://www.dialogic.com/support/. Although this tech note was written for a system release, the information also applies to HMP releases.
- After the templates have been defined in the CONFIG file, you must download this data to the firmware by starting or initializing HMP software. All templates must exist in the firmware before they can be used at runtime in the application.
- Allow sufficient space in the DX_CAP structure for runtime specification of all templates by their template IDs.
- Once the templates are stored in the firmware, use **dx_dial( )** and set the ca_pamd_qtemp field in DX_CAP to the last byte of the template ID to be used.

1.7.3 Restrictions and Limitations

The following restrictions and limitations apply to the use of this PAMD and PVD qualification template feature:

- A maximum of five templates for PAMD and five templates for PVD can be defined.
- PAMD and PVD are mutually exclusive for a particular **dx_dial( )** call. Only one template ID can be specified at a time.
- The template ID is available per call. Once a particular **dx_dial( )** call succeeds, the user requested template ID is no longer stored in the firmware.

1.7.4 Documentation

The online bookshelf provided with Dialogic® Host Media Processing Software 2.0WIN contains information about all release features including features for application development, configuration, administration, and diagnostics.

For more information about the Dialogic® Voice API, see the following documents:

- *Voice API for HMP Programming Guide*
- *Voice API for HMP Library Reference*

For more information about configuration, see the following document:

- *Dialogic® Digital Network Interface Boards Configuration Guide*

1.8 Support for Advanced Network Services

This Service Update verifies that Dialogic® HMP can inter operate with Advanced Network Services (ANS) teaming software. Teaming allows a customer to group multiple physical network adapters (i.e., NIC interfaces) into *virtual* adapters. Establishing this kind of association between adapters makes it possible to support fault tolerance, load balancing, and virtual LAN (VLAN) tagging. Certifying that Dialogic® HMP can operate in conjunction with ANS provides customers valuable tools to improve network reliability.

This section provides an overview of ANS Adapter Teaming. For a more complete description please see:

http://www.dialogic.com/support/helpweb/

1.8.1 ANS Drivers

To determine if a server's network adapter has the ANS software and drivers installed, select a LAN and view the corresponding LAN Properties window. Then select an adapter and review its NIC Properties window, Teaming tab. If Teaming is enabled, it will be apparent.

ANS requires that servers have the appropriate ANS drivers installed. The latest driver version for each adapter type is available at:

http://www.dialogic.com/support/helpweb/

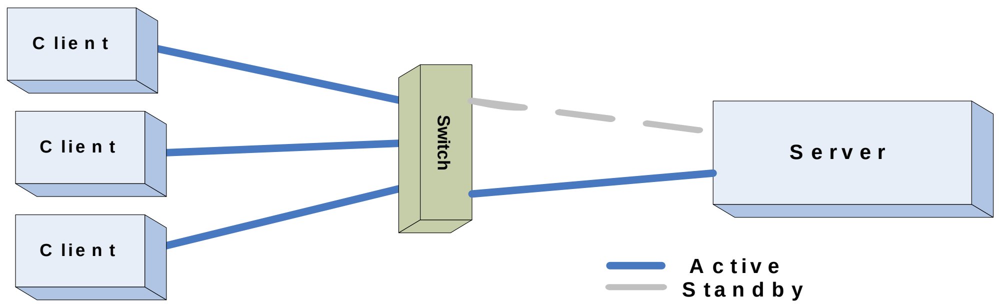
1.8.2 Team Roles

ANS supports teams consisting of from one to eight NIC interfaces, but this feature is limited to two-member teams. In a two member team, one interface serves as the *primary* network interface and the other serves in a *secondary*, standby role. Both fail-over protection and load balancing are supported by assigning primary and secondary team roles.

1.8.3 Fault Tolerance

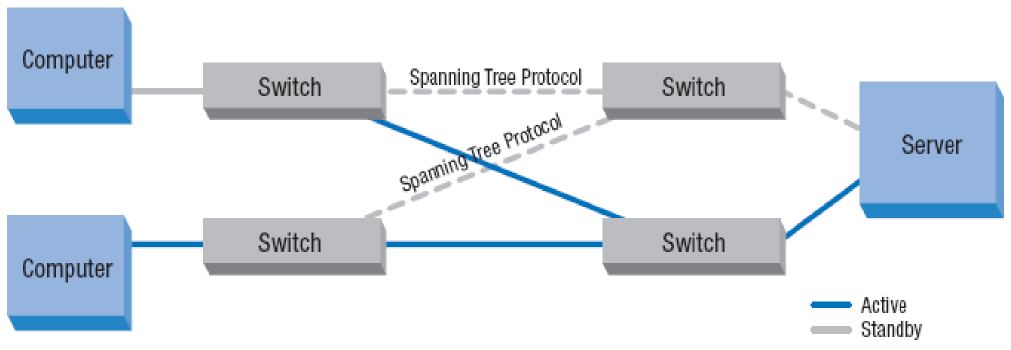
Failure protection is afforded through two types of fault tolerance, *Adapter Fault Tolerance* (AFT) and *Switch Fault Tolerance* (SFT). AFT is the default mode when a team is created. When AFT is operating, the secondary adapter automatically carries traffic when there is a connection failure of any type (e.g., cable, adapter, port, link partner) on the primary. ANS passes the failed primary adapter MAC (Media Access Control) address and Layer 3 (IP) address to the secondary adapter to make this possible. In AFT, all adapters are connected to the same switch or hub, as in the following diagram.

Adapter Fault Tolerance



Switch Fault Tolerance, in contrast, supports a fail-over relationship between two adapters when each adapter is connected to a separate switch. In SFT, if one adapter, its cabling or the switch fails, the other adapter takes over, as in the following diagram.

Switch Fault Tolerance



ANS provides fault tolerance for different network architectures, with both single and multiple switch connections supported.

1.8.4 Load Balancing

Load balancing distributes the transmission and reception traffic among network adapters. This helps to maximize performance and improve reliability. Two types of load balancing are supported, Adaptive Load Balancing (ALB) and Receive Load Balancing (RLB). ALB supports transmission over from two to eight ports. Transmission to multiple destination addresses is supported and ALB incorporates Adapter Fault Tolerance. Receive Load Balancing is a sub feature of ALB and can only work in conjunction with it. RLB allows reception of network traffic on from two to eight ports from multiple client addresses. However, RLB only works with TCP/IP-based traffic.

Note: TCP/IP must be specified as the transport protocol in Global Call before RLB configuration can be run.

For instructions on how to set TCP/IP as the transport protocol, go to *Configuring TCP Transport* in the *Dialogic®Global Call for IP in Host Media Processing Technology Guide*.

1.9 Enhancements to Runtime Trace Facility (RTF) Logging

The Service Update provides enhancements to Runtime Trace Facility (RTF) logging. The RTF tool provides a mechanism for tracing the execution path of Dialogic® runtime libraries. The trace information can be captured in a log file or sent to a system-specific debug stream (e.g., debug console on Windows). The resulting log file/debug stream output helps troubleshoot runtime issues for applications that are built with Dialogic® software.

With the enhanced logging in the Service Update, logging consistency is maintained across all host libraries. There is now a standard set of labels in the RTF configuration file that all modules use. In addition, the logging messages have been improved.

1.9.1 New Labels in the RTF Configuration File

The RTF tool obtains trace control settings and output formatting from an RTF configuration file (*RtfConfigWin.xml*). Each runtime library has several levels of tracing that can be dynamically enabled/disabled by editing the RTF configuration file while your application runs, and entering rtftrace -start and rtftrace -stop.

With the Service Update, there is now a standard set of labels that all modules use in the RTF configuration file. The standard labels are easier to read and follow. The labels are are APPL, INTF, INFO, DEBG, EXCE, WARN, ERR1, and ERR2. The labels for Global Call, DCB, and MSI have been changed to follow this standard; the old and new labels are shown below. The new logging format for these libraries is now consistent with other host libraries such as the DM3 call control/voice/dti libraries.

New Labels	Old Labels

<!-- LIBGC -->
<Module name=”gc” state = “1”>
  <MLabel name=”DEBG” state = “0”/>
  <MLabel name=”INFO” state = “0”/>
  <MLabel name=”WARN” state = “0”/>
  <MLabel name=”APPL” state = “0”/>
  <MLabel name=”EXCE” state = “1”/>
  <MLabel name=”ERR1” state = “1”/>
</Module>
	<!-- LIBGC -->
	  <Module name=”libgc.dll” state = “1”>
	  <MLabel name=”Info” state = “0”/>
	  <MLabel name=”Warning” state = “0”/>
	  <MLabel name=”Debug” state = “0”/>
	  <MLabel name=”Alert” state = “0”/>
	</Module>

<!-- DM3 DCB-->
<Module name=”dm3dcb” state = “1”>
  <MLabel name=”APPL” state = “0”/>
  <MLabel name=”INTF” state = “0”/>
  <MLabel name=”DEBG” state = “0”/>
  <MLabel name=”INFO” state = “0”/>
  <MLabel name=”WARN” state = “0”/>
  <MLabel name=”EXCE” state = “1”/>
  <MLabel name=”ERR1” state = “1”/>
  <MLabel name=”ERR2” state = “1”/>
</Module>
	<!-- DM3 DCB-->
	<Module name=”libDm3Dcb” state = “1”>
	  <MLabel name=”Internal” state = “0”/>
	  <MLabel name=”Warning” state = “0”/>
	  <MLabel name=”Info” state = “0”/>
	  <MLabel name=”External_Event” state = “0”/>
	  <MLabel name=”External” state = “0”/>
	</Module>

<!-- MSI DM3 -->
<Module name=”dm3msi” state = “1”>
  <MLabel name=”APPL” state = “0”/>
  <MLabel name=”INTF” state = “0”/>
  <MLabel name=”DEBG” state = “0”/>
  <MLabel name=”INFO” state = “0”/>
  <MLabel name=”WARN” state = “0”/>
  <MLabel name=”EXCE” state = “1”/>
  <MLabel name=”ERR1” state = “1”/>
  <MLabel name=”ERR2” state = “1”/>
</Module>
	!-- MSI DM3 -->
	<Module name=”libDm3Msi” state = “1”>
	  <MLabel name=”Internal_Entry” state = “0”/>
	  <MLabel name=”Internal_Exit” state = “0”/>
	  <MLabel name=”Warning” state = “0”/>
	  <MLabel name=”Info” state = “0”/>
	  <MLabel name=”External_Event” state = “0”/>
	  <MLabel name=”External_Entry” state = “0”/>
	  <MLabel name=”External_Exit” state = “0”/>
	</Module>

To enable full tracing for a module, set all labels to 1. Default logging has errors and exceptions enabled.

1.10 Determining the Capabilities Supported by an Active Dialogic® HMP Software License

With this feature, a new function, **ipm_GetCapabilities( )**, is added. The new function provides a mechanism to list capabilities of a specified type for an active license. The function returns the number of capabilities (for example, the number of coders) and descriptive information. The function is supported in synchronous mode only.

ipm_GetCapabilities( )

Name:	ipm_GetCapabilities(a_nDeviceHandle, a_CapType, a_num, a_CapabilitiesArray[], a_usMode);	
Inputs:	int a_nDeviceHandle	• IP Media device handle
	eCAPABILITY_TYPE a_CapType	• capability type to be retrieved
	unsigned int a_num	• number of entries in the capability array
	IPM_CAPABILITIES a_CapabilitiesArray[]	• capability array
	unsigned short a_usMode	• async or sync mode setting
Returns:	number of capabilities available -1 on failure	
Includes:	srllib.h ipmlib.h	
Category:	Media Session	
Mode:	synchronous	

■ Description

The **ipm_GetCapabilities( )** function returns the number of capabilities of the specified type (for example, coders) and details of each capability supported by an active HMP software license. The number of capabilities available may be greater than the number specified by the **a_num** input parameter, therefore the following rules apply:

- If **a_num** is zero and/or **a_CapabilitiesArray[]** is NULL, this function returns only the number of capabilities available; no capability detail is retrieved.
- If **a_num** is larger than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of all capabilities and the remaining allocated memory is unused.
- If **a_num** is smaller than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of **a_num** capabilities (that is, as many as will fit); details of the remaining capabilities are not retrieved.

Parameter	Description
n_DeviceHandle	handle of the IP Media device
a_CapType	capability type, for example CAPABILITY_CODERLIST
a_num	the number of entries in the capability array
a_CapabilitiesArray[]	the capability array
a_usMode	operation mode Set to EV_SYNC for synchronous execution

The datatype for the **a_CapabilitiesArray[]** parameter is a union, IPM_CAPABILITIES, which is defined as follows:

```
typedef struct ipm_capabilities_tag
{
   unsigned int version;
   union
   {
      IPM_CODER_INFO Coder;
      // Future types here.
   };
}IPM_CAPABILITIES;
```

In this union, the IPM_CODER_INFO data structure provides coder details such as coder type, frame size, number of frames per packet, VAD enable/disable information and payload-related information.

The datatype for the **a_CapType** parameter is eCAPABILITY_TYPE, an enumeration that is defined as follows:

```
enum CAPABILTIY_TYPE
{
   CAPABILITY_CODERLIST;
}
```

The **ipm_GetCapabilities( )** function is supported in synchronous mode only. If asynchronous mode (a_usMode = EV_ASYNC) is specified, an error is generated.

■ Cautions

None.

■ Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
  Invalid parameter

EIPM_INTERNAL
  Internal error

EIPM_INV_MODE
  Invalid mode

EIPM_INV_STATE
  Invalid state. Initial command did not complete before another function call was made.

EIPM_ERROR
  System error

■ Example

In this example, the first **ipm_GetCapabilities( )** call retrieves only number of capabilities available (count). That number is then used to allocate the right amount of memory and retrieve details of all the capabilities.

```
#include <ipmlib.h>

// Code follows:

unsigned int count;
IPM_CODER_INFO *caps
int i;

unsigned int count=ipm_GetCapabilties(dev,CAPABILITY_CODERLIST,0,NULL,SYNC);
caps=(IPM_CAPABILITIES *malloc(sizeof(IPM_CAPABILITIES)*count);


// check for memory error here

count=ipm_GetCapabilties(dev,CAPABILITY_CODERLIST,count,caps,SYNC);

for (i=0;i<count;i++)
{
    printf("RFC 1890 Coder Type %ui supported\n",caps[i].Coder.unCoderPayloadType);
}


// Free coder list here
free(caps);
```

■ See Also

None.

1.11 Dynamically Retrieving and Modifying Selected Protocol Parameters

With this Service Update, the ability to dynamically retrieve or modify certain protocol-specific parameter values stored by the Dialogic® Digital Network Interface board firmware is provided. The boards that support this feature are:

- DNI300TEPHMP
- DNI601TEPHMP
- DNI1200TEPHMP

1.11.1 Feature Description

This feature allows a user to dynamically (at runtime) retrieve and/or modify the following parameter values:

- Protocol ID
- CAS signal definitions
- CDP variable values
- Line type (E1_CRC, D4, ESF) and coding (B8ZS, HDB3, AMI) for a trunk
- Protocol for a trunk

Some typical use cases for this feature are as follows:

- When a new system is configured and then provisioned by a new carrier, protocol parameters, such as WINK settings, need to be tweaked before a call can be placed using the new switch. Currently, the configuration file has to be manually edited for every change and the firmware re-downloaded for the changes to take effect. The provision of this API to perform these changes at runtime alleviates the need to manually edit configuration files and subsequently re-download the firmware.
- When using ISDN protocols, the ability to dynamically determine the protocol running on a particular span is important in determining whether features such as Two-B Call Transfer (TBCT) or Overlapped Sending can be supported.
- The ability to change certain line parameters at runtime based on fluctuations in carrier provisioning (for example, CRC settings) or depending on whether a site is connecting with the Central Office or a Tie Line to a PBX (for example, User/Network settings).

This feature is implemented using the Global Call Run Time Configuration Management (RTCM) facility, which uses the Global Call **gc_GetConfigData( )**, **gc_SetConfigData( )**, and **gc_QueryConfigData( )** functions. For general information on the operation of the Global Call RTCM facility, see the *Global Call API Programming Guide*. Details on how to dynamically configure the parameter types mentioned above are provided in the sections following.

1.11.1.1 Prerequisites for Feature Use

Creating a dm3enum.cfg File

Before this feature can be used, it must be enabled. To enable the feature, a new configuration file must be created. The name of the configuration file is *dm3enum.cfg* and it must be stored in the *Dialogic\cfg* directory. The *dm3enum.cfg* file determines the boards on which this dynamic protocol configuration feature is to be enabled.

The syntax of the commands that can be included in a *dm3enum.cfg* file are:

board <n>
    Specifies a logical board (<n>) on which this feature is to be enabled.

board (startBd endBd)
Specifies a range of boards on which this feature is to be enabled.

Notes:1. The “board” command word can be abbreviated to “b”.

2. The startBd value must be less than the endBd value.

Some examples of commands that can be included in a *dm3enum.cfg* file are:

```
board 0
b 1
board (1 3)
b (1 3)
```

The feature generates a number of log files in the *Dialogic\log* directory:

Dm3enumerate.log
Log file for application that “kicks off” dynamic protocol configuration enablement

Protocol_enumerate_board_#.log
Log file of actual dynamic protocol configuration enablement process for boards that use non-PDK protocols

Pdkenumerate.log_board#
Log file of actual dynamic protocol configuration enablement process for boards that use PDK protocols

Enabling Protocol Configuration

To enable protocol configuration for PDK protocols, after running the `pdkmanagersetup add` command, run the following command:

```
pdkmanagerregsetup enumerate
```

Then,

- Download the board firmware
- Run the `devmapdump` utility to check that the protocol information has been loaded (search for CDP_ and CAS_ parameters). If this is not the case, run the `dm3enumerate` utility to load the protocol information manually after each firmware download.

To enable protocol configuration for non-PDK protocols, run the `dm3enumerate` utility to load protocol name information.

1.11.1.2 Retrieving a Protocol ID

DM3 protocol names have the format “lb#pv#:Variant_Name”, where:

- lb# is the logical board ID on a physical DM3 board
- pv# is the protocol variant ID

Some examples are:

- "lb1pv1:pdk_us_mf_io" - A PDK protocol that is the first protocol variant on logical board 1
- "lb2pv1:isdn_net5" - An ISDN Net5 protocol that is the first protocol variant on logical board 2
- "lb0pv5:analog_loop_fxs"- An analog protocol that is the fifth protocol variant on logical board 0

Note: All characters in protocol names are lowercase.

The protocol ID is assigned by Global Call and the user must obtain the protocol ID prior to accessing any protocol-related data.

The protocol name and ID for a DM3 board can be obtained by calling the **gc_GetConfigData( )** function on an opened time slot device handle with the following parameter values:

- **target_type** = GCTGT_GCLIB_CHAN
- **target_id** = the line device handle
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )** for protocol ID and **gc_util_insert_parm_ref( )** for protocol name.
- **time_out** = time interval (in seconds) during which data must be retrieved. If the interval is exceeded, the retrieve request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution. EV_SYNC mode is recommended.

Note: Only time slot objects support the retrieval of the protocol ID and name.

See Section 1.11.1.5, "Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values", on page 47, specifically the **ObtainProtocolIDAndName( )** function, for example code that demonstrates how to retrieve the protocol ID and name.

If the protocol name is known, the protocol ID can be obtained by calling the **gc_QueryConfigData( )** function with the following parameter values:

- **target_type** = GCTGT_GCLIB_SYSTEM
- **target_id** = GC_LIB
- **source_datap** = GC_PARM parameter pointer for storing the protocol name (input)
- **query_id** = Query ID, in this case, GCQUERY_PROTOCOL_NAME_TO_ID
- **response_datap** = GC_PARM parameter pointer for storing the protocol ID (output)

1.11.1.3 Retrieving or Modifying CAS Signal Definitions

This feature enables the user to dynamically retrieve or modify CAS signal definitions. Before the CAS signal definition can be retrieved or modified, the {Set ID:Parm ID} pair that identifies the signal in the firmware must be retrieved. The datatype of the corresponding parameter value must also be retrieved. The following sections describe the operations relating to CAS signal definitions that can be performed.

Obtaining the {Set ID:Parm ID} Pair for a CAS Signal

Each CAS parameter in a DM3 PDK protocol has a unique {set ID:parm ID} pair, in which the **set ID** represents the component that contains the parameter and **parm ID** represents an internal ID within that component. The set ID is one of a predefined set of values in the *dm3cc_parm.h* file, and the parm ID is assigned by the DM3 firmware at download time. For example, the CAS_ANSWER parameter (which defines a CAS signal) is contained in the CAS component identified by the PRSET_CAS_SIGNAL set ID with the parm ID being assigned internally by the firmware.

Before dynamically retrieving or modifying the value of a CAS parameter in the DM3 firmware, the user must call the **gc_QueryConfigData( )** function to obtain the {setID, parmID} pair of the CAS parameter using the parameter name obtained from the CDP file.

The **gc_QueryConfigData( )** function is called with the following parameter values:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **source_datap** = GC_PARM parameter pointer for storing input CAS parameter name
- **query_id** = Query ID, in this case, GCQUERY_PARM_NAME_TO_ID
- **response_datap** = GC_PARM parameter pointer for storing output {set ID, parm ID} and value type

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **QueryParmID( )** function, for example code that demonstrates how to retrieve the {Set ID:Parm ID} pair for a CAS signal.

Note: Obtaining the {Set ID:Parm ID} pair is a prerequisite to retrieving the definition of a CAS signal or redefining a CAS signal.

Retrieving a CAS Signal Definition

The **gc_GetConfigData( )** function can be used to retrieve the value of CAS parameters in the DM3 firmware. Function parameter values to use in this context are:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the **gc_util_insert_parm_ref( )** utility function for CAS signal
- **time_out** = time interval (in seconds) during which parameter value must be retrieved. If the interval is exceeded, the retrieve request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **request_idp** = pointer to the location for storing the request ID, output from Global Call

- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution. EV_ASYNC mode is recommended.

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **GetCASSignalDef( )** function, for example code that demonstrates how to retrieve the definition of a CAS signal, in this case the CAS_WINKREV signal.

Setting a CAS Signal Definition

The **gc_SetConfigData( )** function with the following parameter values can be used to set a new definition for a CAS signal in the DM3 firmware:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_ref( )** for the CAS signal
- **time_out** = time interval (in seconds) during which the parameter value must be updated. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = Ignored for DM3 PDK protocols
- **request_idp** = pointer to the location for storing the request ID, output from Global Call
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution. EV_ASYNC mode is recommended.

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **GetCASSignalDef( )** function, for example code that demonstrates how to change the definition of a CAS signal, in this case the CAS_WINKREV signal.

1.11.1.4 Retrieving or Modifying CDP Variable Values

This feature enables the user to dynamically retrieve or modify parameter values defined in DM3 PDK protocol Country Dependent Parameter (CDP) files. Before the CDP variable value can be retrieved or modified, the {Set ID:Parm ID} pair that identifies the CDP variable in the firmware must be retrieved. The datatype of the corresponding CDP variable value must also be retrieved. The following sections describe the operations relating to CDP variable values that can be performed.

Obtaining the {Set ID:Parm ID} Pair for a CDP Variable

Each CDP variable in a DM3 PDK protocol has a unique {set ID:parm ID} pair, in which the **set ID** represents the component that contains the parameter and **parm ID** represents an internal ID within that component. The set ID is one of a predefined set of values in the *dm3cc_parm.h* file, and the parm ID is assigned by the DM3 firmware at download time.

Before dynamically retrieving or modifying the value of a CDP variable in the DM3 firmware, the user must call the **gc_QueryConfigData( )** function to obtain the {setID, parmID} pair of the CDP variable using the parameter name obtained from the CDP file.

The **gc_QueryConfigData( )** function is called with the following parameter values:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **source_datap** = GC_PARM parameter pointer for storing the input CDP variable name
- **query_id** = Query ID, in this case, GCQUERY_PARM_NAME_TO_ID
- **response_datap** = GC_PARM parameter pointer for storing the output {set ID, parm ID} and value type

Note: Obtaining the {Set ID:Parm ID} pair is a prerequisite to retrieving or changing the value of a CDP variable.

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **QueryParmID( )** function, for example code that demonstrates how to retrieve the {Set ID:Parm ID} pair for a CDP variable.

Getting the Current Values of Multiple CDP Variables

The **gc_GetConfigData( )** function can be used to retrieve the value of a CDP variable in the DM3 firmware. Function parameter values to use in this context are:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )** for CDP integer value and **gc_util_insert_parm_ref( )** for CDP string value.
- **time_out** = time interval (in seconds) during which the parameter value must be retrieved. If the interval is exceeded, the retrieve request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **request_idp** = pointer to the location for storing the request ID, output from Global Call
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution. EV_ASYNC mode is recommended.

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **GetCDPVarParms( )** function, for example code that demonstrates how to get the current values of multiple CDP variables.

Setting New Values for Multiple CDP Variables

The **gc_SetConfigData( )** function can be used to set new values for multiple CDP variables in the DM3 firmware. Function parameter values to use in this context are:

- **target_type** = GCTGT_PROTOCOL_SYSTEM
- **target_id** = PDK Protocol ID
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )** for the CDP integer value and **gc_util_insert_parm_ref( )** for the CDP string value.
- **time_out** = time interval (in seconds) during which the parameter value must be updated. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = ignored for DM3 PDK protocol parameters
- **request_idp** = pointer to the location for storing the request ID, output from Global Call
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution. EV_ASYNC mode is recommended.

See Section 1.11.1.5, “Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values”, on page 47, specifically the **SetCDPVarParms( )** function, for example code that demonstrates how to set new values of multiple CDP variables.

1.11.1.5 Sample Code for Getting and Setting CAS Signal Definitions and CDP Variable Values

```
/* Dialogic Header Files */
#include <gcip.h>
#include <gclib.h>
#include <gcisdn.h>
#include <srllib.h>
#include <dm3cc_parm.h>


int ObtainProtocolIDAndName(LINEDEV a_GCLineDevH, char *a_pProtName, long *a_pProtID)
{
    GC_PARM_BLK * t_pParmBlk = NULL;
    GC_PARM_DATA * t_pParmData = NULL;
    char * t_ProtName[20];
    long t_RequestID = 0;
    int t_result;

    /* Reserve the space for protocol ID */
    *a_pProtID = 0;
    gc_util_insert_parm_val(&t_pParmBlk, GCSET_PROTOCOL, GCPARM_PROTOCOL_ID, sizeof(long),
                            *a_pProtID);
    /* Reserve the space for protocol Name */

    gc_util_insert_parm_ref(&t_pParmBlk, GCSET_PROTOCOL, GCPARM_PROTOCOL_NAME,
                            sizeof(t_ProtName), t_ProtName);
    /* Since the protocol information has already been stored in GC library during gc_OpenEx(),
        it is recommended to call gc_GetConfigData() in SYNC mode */
    t_result = gc_GetConfigData(GCTGT_GCLIB_CHAN, a_GCLineDevH, t_pParmBlk, 0, & t_RequestID,
                                EV_SYNC);
    if (t_result)
    {
        /* Process the error */
        gc_util_delete_parm_blk(t_pParmBlk);
        return t_result;
    }
    /* Obtain the protocol ID */
    t_pParmData = gc_util_find_parm(t_pParmBlk, GCSET_PROTOCOL, GCPARM_PROTOCOL_ID);
    if (NULL != t_pParmData)
    {
        memcpy(a_pProtID, t_pParmData->value_buf, t_pParmData->value_size);
    }
    /* Obtain the protocol Name */
    t_pParmData = gc_util_find_parm(t_pParmBlk, GCSET_PROTOCOL, GCPARM_PROTOCOL_NAME);
    if (NULL != t_pParmData)
    {
        strcpy(a_pProtName, (const char*)t_pParmData->value_buf);
    }
    printf("ObtainProtocolIDAndName(linedev:%d, protocol_id:%d, protocol_name:%s)",
            a_GCLineDevH, *a_pProtID, a_pProtName);
    gc_util_delete_parm_blk(t_pParmBlk);
    return t_result;
}


int QueryParmID(long a_PDKProtocolID, char *a_pParmName, unsigned short * a_pSetID,
                unsigned short * a_pParmID, unsigned char * a_pValType)
{
    GC_PARM t_SourceData;
    GC_PARM t_RespData;
    GC_PARM_ID t_ParmIDBlk;
    int t_result = 0;

    /* Pass the CDP name, which is defined in CDP file, e.g., "CAS_WINKRCV" or "CDP_ANI_ENABLED"
       in pdk_us_mf_io.cdp */
    t_SourceData.paddress = a_pParmName;
    memset(&t_ParmIDBlk, '0', sizeof(GC_PARM_ID));
    t_RespData.pstruct = & t_ParmIDBlk;
    t_result = gc_QueryConfigData(GCTGT_PROTOCOL_SYSTEM, a_PDKProtocolID, &t_SourceData,
                                  GCQUERY_PARM_NAME_TO_ID, &t_RespData);
    if (t_result)
    {
        /* Process the error */
        *a_pSetID = 0;
        *a_pParmID = 0;
        *a_pValType = 0;
        printf("gc_QueryConfigData(parm:%s) failed on protocol:%d", a_pParmName,
                a_PDKProtocolID);
    }
    else
    {
        *a_pSetID = t_ParmIDBlk.set_ID;
        *a_pParmID =  t_ParmIDBlk.parm_ID;
        *a_pValType = t_ParmIDBlk.value_type;
        printf("gc_QueryConfigData(parm:%s) succeed with  {setID:0x%x, parmID:0x%x, valType:%d}
               on protocol:%d",
               a_pParmName, *a_pSetID, *a_pParmID, *a_pValType, a_PDKProtocolID);
    }
    return t_result;
}

int SetCASSignalDef(long a_PDKProtocolID)
{
    GC_PARM_BLK * t_pParmBlk = NULL;
    unsigned short t_SetID;
    unsigned short t_ParmID;
    unsigned char t_ValType;
    long t_RequestID = 0;
    int t_result = 0;
    GC_CASPROT_TRANS t_CasTrans;
    GC_CASPROT_PULSE t_CasPulse = {"00xx", "11xx", 50, 62, 0, 80, 20, 250, 300};
    GC_CASPROT_TRAIN t_CasTrain;
    /* Find the {setID, parmID, DataType} of CAS_WINKRCV for pdk_us_mf_io */
    t_result = QueryParmID(a_PDKProtocolID, "CAS_WINKRCV", &t_SetID, &t_ParmID, &t_ValType);
    if (t_result)
    {
        /* Process the error */
        return t_result;
    }
    /* Insert new definition for CAS signals, dependent on the signal type */
    switch (t_ValType)
    {
        case GC_VALUE_CAS_TRANS:
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_TRANS),
                                    &t_CasTrans);
        break;
        case GC_VALUE_CAS_PULSE:
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_PULSE),
                                    &t_CasPulse);
        break;
        case GC_VALUE_CAS_TRAIN:
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_TRAIN),
                                    &t_CasTrain);
        break;
        default:
        /* Process the error here */
        return -1;
        break;
    }
    /* Set the CAS_WINKRCV with new value */
    t_result = gc_SetConfigData(GCTGT_PROTOCOL_SYSTEM, a_PDKProtocolID, t_pParmBlk, 0,
                                GCUPDATE_IMMEDIATE, &t_RequestID, EV_ASYNC);
    if (t_result)
    {
        /* Process the error */
        gc_util_delete_parm_blk(t_pParmBlk);
        return t_result;
    }
    gc_util_delete_parm_blk(t_pParmBlk);
    return t_result;
}

int GetCASSignalDef(long a_PDKProtocolID)
{
    GC_PARM_BLK * t_pParmBlk = NULL;
    unsigned short t_SetID;
    unsigned short t_ParmID;
    unsigned char t_ValType;
    long t_RequestID = 0;
    int t_result = 0;
    GC_CASPROT_TRANS t_CasTrans;
    GC_CASPROT_PULSE t_CasPulse;
    GC_CASPROT_TRAIN t_CasTrain;
    /* Find the {setID, parmID, dataType} of CAS_WINKRCV for pdk_us_mf_io */
    t_result = QueryParmID(a_PDKProtocolID, "CAS_WINKRCV", &t_SetID, &t_ParmID, &t_ValType);
    if (t_result)
    {
        /* Process the error */
        return t_result;
    }
    /* Insert memory space for storing definition for CAS signals, dependent on the signal type
    */
    switch (t_ValType)
    {
        case GC_VALUE_CAS_TRANS:
            memset( &t_CasPulse, 0, sizeof(GC_CASPROT_TRANS) );
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_TRANS),
                                    &t_CasTrans);
        break;
        case GC_VALUE_CAS_PULSE:
            memset( &t_CasPulse, 0, sizeof(GC_CASPROT_PULSE) );
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_PULSE),
                                    &t_CasPulse);
        break;
        case GC_VALUE_CAS_TRAIN:
            memset( &t_CasPulse, 0, sizeof(GC_CASPROT_TRAIN) );
            gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, sizeof(GC_CASPROT_TRAIN),
                                    &t_CasTrain);
        break;
        default:
            /* Process the error here */
            return -1;
        break;
    }
    /* Get the CAS_WINKRCV with new value */
    t_result = gc_GetConfigData(GCTGT_PROTOCOL_SYSTEM, a_PDKProtocolID, t_pParmBlk, 0,
                                &t_RequestID, EV_ASYNC);
    if (t_result)
    {
        /* Process the error */
        gc_util_delete_parm_blk(t_pParmBlk);
        return t_result;
    }
    gc_util_delete_parm_blk(t_pParmBlk);
    return t_result;
}



typedef struct {
    char        name[50];
    int         type;
    void *      valuep;
} CDP_PARM;

int GetCDPVarParms(long a_PDKProtocolID, int a_NumParms, CDP_PARM * a_CDPVarParms, long *
a_pRequestID)
{
    GC_PARM_BLK * t_pParmBlk = NULL;
    unsigned short t_SetID;
    unsigned short t_ParmID;
    unsigned char t_ValType = 0;
    int t_result = 0;
    int index1 = 0;

    if (!a_PDKProtocolID)
    {
        /* Process the error */
        return -1;
    }
    if (!a_CDPVarParms)
    {
        /* Process the error */
        return -1;
    }
    /* Support retrieving multiple CDP variables in a single gc_GetConfigData() function call */
    for (index1 = 0; index1 < a_NumParms; index1 ++)
    {
        /* Find the {setID, parmID, valueType} of each CDP variable by its name: e.g.,
           "CDP_ANI_ENABLED" in pdk_ar_r2_io.cdp */
        t_result = QueryParmID(a_PDKProtocolID, a_CDPVarParms[index1].name, &t_SetID, &t_ParmID,
                               &t_ValType);
        if (t_result)
        {
            /* Process the error */
            gc_util_delete_parm_blk(t_pParmBlk);
            return t_result;
        }
        if (t_SetID != PRSET_TSC_VARIABLE)
        {
            /* Not a CDP variable parameter */
            gc_util_delete_parm_blk(t_pParmBlk);
            return -1;
        }
        /* Insert new definition for CDP variable signals, dependent on the value data type */
        switch (t_ValType)
        {
            case GC_VALUE_SHORT:
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(unsigned short),
                                        0);
            break;
            case GC_VALUE_STRING:
                gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, 30, "");
            break;
            case GC_VALUE_ULONG:
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(unsigned long),
                                        0);
            break;
            case GC_VALUE_UCHAR:
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(unsigned char),
                                        0);
            break;
            default:
                /* Process the error here */
                printf("!!!!Invalid value type for protocolID:%d to CDP variable(name:%s,
                        set_id:0x%x, parm_id:0x%x, valtype:%d)",
                        a_PDKProtocolID, a_CDPVarParms[index1].name, t_SetID, t_ParmID,
                        t_ValType);
                gc_util_delete_parm_blk(t_pParmBlk);
                return -1;
            break;
        }
    }
    /* Get the values of multiple CDP variables */
    *a_pRequestID = 0;
    t_result = gc_GetConfigData(GCTGT_PROTOCOL_SYSTEM, a_PDKProtocolID, t_pParmBlk, 0,
                                a_pRequestID, EV_ASYNC);
    if (t_result)
    {
        /* Process the error */
        printf("gc_GetConfigData(protocol_id:%d) failed on setting CDP parameters()",
                a_PDKProtocolID);
        *a_pRequestID = 0;
    }
    else
    {
        printf("gc_GetConfigData(protocol_id:%d, req_id:0x%x) succeed on setting CDP
               parameters",
               a_PDKProtocolID, *a_pRequestID);
    }
    gc_util_delete_parm_blk(t_pParmBlk);
    return t_result;
}

int SetCDPVarParms(long a_PDKProtocolID, int a_NumParms, CDP_PARM * a_CDPVarParms, long *
a_pRequestID)
{
    GC_PARM_BLK * t_pParmBlk = NULL;
    unsigned short t_SetID;
    unsigned short t_ParmID;
    unsigned char t_ValType = 0;
    int t_result = 0;
    int t_IntVal = 0;
    unsigned long t_ULongVal = 0;
    unsigned char t_UCharVal = 0;
    unsigned char t_StrSize = 0;
    int index1 = 0;

    if (!a_PDKProtocolID)
    {
        /* Process the error */
        return -1;
    }
    if (!a_CDPVarParms)
    {
        /* Process the error */
        return -1;
    }
    /* Support setting multiple CDP variables in a single gc_SetConfigData() function call */
    for (index1 = 0; index1 < a_NumParms; index1 ++)
    {
        /* Find the {setID, parmID, valueType} of each CDP variable by its name: e.g.,
           "CDP_ANI_ENABLED" in pdk_ar_r2_io.cdp */
        t_result = QueryParmID(a_PDKProtocolID, a_CDPVarParms[index1].name, &t_SetID, &t_ParmID,
                               &t_ValType);
        if (t_result)
        {
            /* Process the error */
            gc_util_delete_parm_blk(t_pParmBlk);
            return t_result;
        }
        if (t_SetID != PRSET_TSC_VARIABLE)
        {
            /* Not a CDP variable parameter */
            gc_util_delete_parm_blk(t_pParmBlk);
            return -1;
        }
        /* Insert new definition for CDP variable signals, dependent on the value data type */
        switch (t_ValType)
        {
            case GC_VALUE_INT:
                t_IntVal = *((int*)a_CDPVarParms[index1].valuep);
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(int), t_IntVal);
                printf("Set Integer Value:%d (0x%x) to parmID:0x%x",
                            t_IntVal, t_IntVal, t_ParmID);
            break;
            case GC_VALUE_STRING:
                t_StrSize = strlen((char *)a_CDPVarParms[index1].valuep) + 1;
                gc_util_insert_parm_ref(&t_pParmBlk, t_SetID, t_ParmID, t_StrSize, (char *)
                                        a_CDPVarParms[index1].valuep);
                printf("Set String Value:%s to parmID:0x%x",
                            (char *) a_CDPVarParms[index1].valuep, t_ParmID);
            break;
            case GC_VALUE_ULONG:
                t_ULongVal = *((unsigned long *)a_CDPVarParms[index1].valuep);
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(unsigned long),
                                        t_ULongVal);
                printf("Set Long Value:%d (0x%x) to parmID:0x%x",
                            t_ULongVal, t_ULongVal, t_ParmID);
            break;
            case GC_VALUE_UCHAR:
                t_UCharVal = *((unsigned char *)a_CDPVarParms[index1].valuep);
                gc_util_insert_parm_val(&t_pParmBlk, t_SetID, t_ParmID, sizeof(unsigned char),
                                        t_UCharVal);
                printf("Set Char Value:%d(0x%x) to parmID:0x%x",
                            t_UCharVal, t_UCharVal, t_ParmID);
            break;
```

```
            default:
                /* Process the error here */
                printf("!!!!Invalid value type for protocolID:%d to CDP variable(name:%s,
                        set_id:0x%x, parm_id:0x%x, valtype:%d)",
                        a_PDKProtocolID, a_CDPVarParms[index1].name, t_SetID, t_ParmID,
                        t_ValType);
                gc_util_delete_parm_blk(t_pParmBlk);
                return -1;
            break;
        }
    }
    /* Set the CDP parameters with new values */
    *a_pRequestID = 0;
     t_result  = gc_SetConfigData(GCTGT_PROTOCOL_SYSTEM, a_PDKProtocolID, t_pParmBlk, 0,
                                  GCUPDATE_IMMEDIATE, a_pRequestID, EV_ASYNC);
    if (t_result)
    {
        /* Process the error */
        printf("gc_SetConfigData(protocol_id:%d) failed on setting CDP parameters()",
                a_PDKProtocolID);
        *a_pRequestID = 0;
    }
    else
    {
        printf("gc_SetConfigData(protocol_id:%d, req_id:0x%x) succeed on setting CDP
                parameters",
            a_PDKProtocolID, *a_pRequestID);
    }
    gc_util_delete_parm_blk(t_pParmBlk);
    return t_result;
}


int ProcessRTCMEvent(unsigned long a_GCEvent, unsigned long a_ReqID, GC_PARM_BLK * a_pParmBlk)
{
    GC_CASPROT_TRANS * t_pCasTrans = NULL;
    GC_CASPROT_PULSE * t_pCasPulse = NULL;
    GC_CASPROT_TRAIN * t_pCasTrain = NULL;
    unsigned char t_UCharVal = 0;
    unsigned short t_UShortVal = 0;
    unsigned long t_ULongVal = 0;
    char * t_StringVal = NULL;
    int t_StrLen = 0;

    /* Obtain the first parameter */
    GC_PARM_DATA * t_pParmData = gc_util_next_parm(a_pParmBlk, NULL);
    while (t_pParmData)
    {
        if (t_pParmData->set_ID == PRSET_CAS_SIGNAL)
        {
            /* This is a CAS signal */
            if (t_pParmData->value_size == sizeof(GC_CASPROT_TRANS) )
            {
                t_pCasTrans = (GC_CASPROT_TRANS *) &t_pParmData->value_buf;
                printf("Obtain CAS Trans signal definition on parmID:0x%x (%s, %s, %d, %d, %d,
                        %d)",
                    t_pParmData->parm_ID, t_pCasTrans->PreTransCode, t_pCasTrans->PostTransCode,
                    t_pCasTrans->PreTransInterval, t_pCasTrans->PostTransInterval,
                    t_pCasTrans->PreTransIntervalNom, t_pCasTrans->PostTransIntervalNom);
            }
            else if (t_pParmData->value_size == sizeof(GC_CASPROT_PULSE) )
            {
                t_pCasPulse = (GC_CASPROT_PULSE *) &t_pParmData->value_buf;
                printf("Obtain CAS Pulse signal definition on parmID:0x%x (%s, %s, %d, %d, %d,
                        %d, %d, %d, %d) ",
                    t_pParmData->parm_ID, t_pCasPulse->OffPulseCode, t_pCasPulse->OnPulseCode,
                    t_pCasPulse->PrePulseInterval, t_pCasPulse->PostPulseInterval,
                    t_pCasPulse->PrePulseIntervalNom, t_pCasPulse->PostPulseIntervalNom,
                    t_pCasPulse->PulseIntervalMin, t_pCasPulse->PulseIntervalNom,
                    t_pCasPulse->PulseIntervalMax);
            }
            else if (t_pParmData->value_size == sizeof(GC_CASPROT_TRAIN) )
            {
                t_pCasTrain = (GC_CASPROT_TRAIN *) &t_pParmData->value_buf;
                printf("Obtain CAS Train signal definition on parmID:0x%x (%s, %s, %d, %d, %d,
                        %d, %d, %d, %d) ",
                    t_pParmData->parm_ID, t_pCasTrain->OffPulseCode, t_pCasTrain->OnPulseCode,
                    t_pCasTrain->PreTrainInterval, t_pCasTrain->PostTrainInterval,
                    t_pCasTrain->PreTrainIntervalNom, t_pCasTrain->PostTrainIntervalNom,
                    t_pCasTrain->PulseIntervalMin, t_pCasTrain->PulseIntervalNom,
                    t_pCasTrain->PulseIntervalMax);
            }
            else
            {
                printf("Error! Incorrect value_size =%d for {setID:0x%x, parmID:0x%x}",
                        t_pParmData->value_size, t_pParmData->set_ID, t_pParmData->parm_ID);
            }
        }
        else if (t_pParmData->set_ID == PRSET_TSC_VARIABLE)
        {
            /* This is a TSC Variable */
            switch (t_pParmData->value_size)
            {
                case 1:
                    /* Unisgned char data */
                    memcpy(&t_UCharVal, &t_pParmData->value_buf,t_pParmData->value_size);
                    printf("Obtain TSC unsigned char value:%d(0x%x) of parmID:0x%x\n",
                                        t_UCharVal, t_UCharVal, t_pParmData->parm_ID);
                    break;
                case 2:
                    /* Unisgned short data */
                    memcpy(&t_UShortVal, &t_pParmData->value_buf,t_pParmData->value_size);
                    printf("Obtain TSC unsigned short value:%d(0x%x) of parmID:0x%x\n",
                        t_UShortVal, t_UShortVal, t_pParmData->parm_ID);
                    break;
                case 4:
                    /* Unisgned long data */
                    memcpy(&t_ULongVal, &t_pParmData->value_buf,t_pParmData->value_size);
                    printf("Obtain TSC integer value:%d(0x%x) of parmID:0x%x",
                            t_ULongVal, t_ULongVal,  t_pParmData->parm_ID);
                    break;
                default:
                {
                    t_StringVal = (char*) t_pParmData->value_buf;
                    t_StrLen = strlen(t_StringVal);
                    if ( t_pParmData->value_size > t_StrLen)
                    {
                        /* String data */
                        printf("Obtain TSC string value:%s(first char: 0x%x) of
                                parmID:0x%x",t_StringVal, t_StringVal[0], t_pParmData->parm_ID);
                    }
                    else
                    {
                        /* Unsupported value size */
                        printf("Unsupported value size:%d for TSC variable parmID:0x%x",
                                t_pParmData->value_size, t_pParmData->parm_ID);
                    }
                }
                break;
            }
        }
        else
        {
            /* Unsupported set ID */
```

```
            printf("Unsupported set_id:0x%x with (parmID:0x%x, value_size:%d) ",
                    t_pParmData->set_ID, t_pParmData->parm_ID, t_pParmData->value_size);
        }
        /* Obtain next parameter */
        t_pParmData = gc_util_next_parm(a_pParmBlk, t_pParmData);
    }
    return 0;
}

struct channel
{
   LINEDEV          LineDev;             /* GlobalCall line device handle */
   char             DevName[50];
   long             ProtocolID;
} port[120];

void process_event()
{
    METAEVENT           metaevent;
    int                 evttype;
    GC_RTCM_EVTDATA *   t_pRtcmEvt = NULL;
    int                 t_Result = 0;
    int                 index = 0;
    struct channel      *pline = NULL;
    char t_ProtocolName[30];
    int t_NumParms = 0;
    int t_RequesID = 0;
    CDP_PARM t_CDPVarParms[3] = {
        {"CDP_IN_WinkStart", GC_VALUE_INT, 0},
        {"CDP_OUT_WinkStart", GC_VALUE_INT, 0},
        {"CDP_OUT_Send_Alerting_After_Dialing", GC_VALUE_INT, 0}
    };


    /* Populate the metaEvent structure */
    if(gc_GetMetaEvent(&metaevent) != GC_SUCCESS)
    {
        printf("gc_GetMetaEvent() failed \n");
       /* Process error */
    }
    /* process GlobalCall events */
    if ((metaevent.flags & GCME_GC_EVENT) == 0)
    {
        printf("Received a non-GC Event 0x%lx\n", metaevent.evttype);
        return;
    }
    evttype = metaevent.evttype;
    if (metaevent.usrattr)
    {
        pline = (struct channel *) metaevent.usrattr;
    }
    switch (evttype)
    {
        case GCEV_UNBLOCKED:
        {
            int t_IntVal = 1;
            t_Result = ObtainProtocolIDAndName(pline->LineDev, t_ProtocolName,
                       &pline->ProtocolID);
            if (t_Result)
            {
                /* Error processs */
            }
            t_NumParms = 3;
            t_CDPVarParms[0].valuep = &t_IntVal;
            t_CDPVarParms[1].valuep = &t_IntVal;
            t_CDPVarParms[2].valuep = &t_IntVal;
            /* Setting new values to CDP variables */
            t_Result = SetCDPVarParms(pline->ProtocolID, t_NumParms, t_CDPVarParms,
                                      &t_RequesID);
            if (t_Result)
            {
                /* Processs error */
            }
        }
            break;
        case GCEV_GETCONFIGDATA:
            t_pRtcmEvt = (GC_RTCM_EVTDATA *) metaevent.evtdatap;
            if (! t_pRtcmEvt || !t_pRtcmEvt->retrieved_parmblkp)
            {
                break;
            }
            printf("Received GCEV_GETCONFIGDATA EVENT on target_type=%d, target_id=0x%x,
                    rquest_id=0x%x",
                    t_pRtcmEvt->target_type, t_pRtcmEvt->target_id, t_pRtcmEvt->request_ID);
            ProcessRTCMEvent(evttype, t_pRtcmEvt->request_ID, t_pRtcmEvt->retrieved_parmblkp);
            break;                                  /* RETURN POINT!!!!! */
        break;
        case GCEV_SETCONFIGDATA:
            t_pRtcmEvt = (GC_RTCM_EVTDATA *) metaevent.evtdatap;
            if (! t_pRtcmEvt)
            {
                break;
            }
            printf("Received GCEV_SETCONFIGDATA EVENT on target_type=%d, target_id=0x%x,
                   rquest_id=0x%x",
                   t_pRtcmEvt->target_type, t_pRtcmEvt->target_id, t_pRtcmEvt->request_ID);
            t_NumParms = 3;
            /* Retrieving existing values from CDP variables */
            t_Result = GetCDPVarParms(t_pRtcmEvt->target_id, t_NumParms, t_CDPVarParms,
                                      &t_RequesID);
            if (t_Result)
            {
                /* Processs error */
            }
        break;
        case GCEV_GETCONFIGDATA_FAIL:
            t_pRtcmEvt = (GC_RTCM_EVTDATA *) metaevent.evtdatap;
            if (! t_pRtcmEvt)
            {
                break;
            }
            printf("Received GCEV_GETCONFIGDATA EVENT_FAIL on target_type=%d, target_id=0x%x,
                   rquest_id=0x%x",
                   t_pRtcmEvt->target_type, t_pRtcmEvt->target_id, t_pRtcmEvt->request_ID);
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            t_pRtcmEvt = (GC_RTCM_EVTDATA *) metaevent.evtdatap;
            if (! t_pRtcmEvt)
            {
                break;
            }
            printf("Received GCEV_SETCONFIGDATA_FAIL EVENT on target_type=%d, target_id=0x%x,
                    rquest_id=0x%x",
                    t_pRtcmEvt->target_type, t_pRtcmEvt->target_id, t_pRtcmEvt->request_ID);
            break;
        default:
            break;
    }
}
```

1.11.1.6 Dynamically Configuring a Trunk

This feature enables the user to perform the following dynamic configuration operations at runtime:

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

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

Setting the Line Type and Coding for a Trunk

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

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

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

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

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_CODING_TYPE
- **data_size** = sizeof(int)

- **data** = One of the following values:
  - Enum_CodingType_AMI - Alternate Mark Inversion
  - Enum_CodingType_B8ZS - Modified AMI used on T1 lines
  - Enum_CodingType_HDB3 - High Density Bipolar of Order 3 used on E1 lines

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

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

The application receives one of the following events:

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

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

```
GC_PARM_BLKP ParmBlkp = NULL;
long id;

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

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

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

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

Specifying the Protocol for a Trunk

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

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

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

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

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

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

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

The application receives one of the following events:

- GCEV_SETCONFIGDATA to indicate that the request to dynamically change the protocol has been successfully initiated.
- GCEV_SETCONFIGDATA_FAIL to indicate that the request to change the protocol has failed. More information is available from the GC_RTCM_EVTDATA structure associated with the event.

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

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

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

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

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

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

Setting the ISDN Protocol User/Network Mode for a Trunk

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

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

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

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

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

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

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

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

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

```
GC_PARM_BLKP ParmBlkp = NULL;
long id;
if(DoWeWantToSetModeInThisSample) {
    int valueMode = 0;             /* 0 for User mode, 1 for Network mode */
    gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_USER_NETWORK, sizeof(int),
                            valueMode);
    gc_SetConfigData(GCTGT_CCLIB_CHAN, ldev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
    gc_util_delete_parm_blk(ParmBlkp);
    }
if (sr_waitevt(-1) >= 0) {
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype()) {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s \n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device %s, Error=%s\n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()),
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevttype(),
                  ATDV_NAMEP(sr_getevtdev()));
            break;
    }

}
```

1.11.2 Extended and New Data Structures

Extension of GC_RTCM_EVTDATA

Two new data fields (target_type and target_id) are appended to the GC_RTCM_EVTDATA data structure defined in the *gclib.h* file. The GC_RTCM_EVTDATA structure is generally associated with Global Call RTCM events (namely, GCEV_SETCONFIGDATA, GCEV_SETCONFIGDATA_FAIL, GCEV_GETCONFIGDATA and GCEV_GETCONFIGDATA_FAIL).

The following shows the extended GC_RTCM_EVTDATA data structure with the new fields shown in **bold** text:

```
typedef struct{
    long            request_ID;          /* The RTCM request ID */
    int             gc_result;           /* GC result value for this event */
    int             cclib_result;        /* CCLib result value for this event */
    int             cclib_ID;            /* CCLib ID for the result */
    char *          additional_msg;      /* Additional message for this event */
    GC_PARM_BLKP    retrieved_parmblkp;  /* Retrieved GC_PARM_BLK -- */
                                         /* used for gc_GetConfigData() in */
                                         /* asynchronous mode */
    int             target_type;         /* Target type */
    long            target_id;           /* Target ID */
} GC_RTCM_EVTDATA, *GC_RTCM_EVTDATAP;
```

The addition of the target_type and target_id fields enables applications to easily identify the DM3 protocol object associated with an event. The change is backward-compatible with usage in current applications. Note that the line_dev and crn accessed by the evtdatap pointer in the METAEVENT structure are zero for DM3 protocol target objects.

The reasoning for adding the target_type, target_id and version fields to the GC_RTCM_EVTDATA data structure is described as follows:

- The values of DM3 protocol parameters are set or retrieved using the **gc_SetConfigData( )** or **gc_GetConfigData( )** functions in asynchronous mode on the protocol target object (type: GCTGT_PROTOCOL_SYSTEM, id: GC Protocol ID). The currently defined Global Call METAEVENT data structure does not provide any information about the target object (that is, the type and ID) when the target object is neither a line device nor a CRN (Call Reference Number).
- This feature is the first introduction of the "protocol target" object to represent the DM3 protocol variant because CDP variable parameters are stored in individual DM3 protocol variants in the CHP component of the DM3 firmware. When the application receives the event for target objects such as DM3 protocols, the linedev and crn fields in the METAEVENT are zero and the associated GC_RTCM_EVTDATA can only provide the request_ID, from which the target object cannot be deduced. Therefore, the application would have to maintain a mapping between request_ID and target object (type and ID).
- For convenience and integration, the GC_RTCM_EVTDATA has been extended to include two new fields, target_type and target_id. This allows the application to easily identify which target object the Global Call RTCM events are associated with.
- The version field has been added to identify the data structure version.
- The new fields are appended to the end of GC_RTCM_EVTDATA to retain backward-compatibility with the existing applications. Also, note that the GC_RTCM_EVTDATA data structure is currently only used by the Global Call library to present the event data to the customer application and is not used by any of the call control libraries.

New Data Structures for CAS Signals

New data structures are defined in the *gclib.h* file that are used by the **gc_SetConfigData( )** and **gc_GetConfigData( )** functions to retrieve/modify the CAS signal definitions associated with a PDK protocol.

As a convenience that enables the user to enter a new CAS signal definition and retrieve the current CAS signal definition, the fields in these data structures strictly follow the same sequence as the CAS signal definitions in the PDK CDP file. Since CAS signal defines in the CDP file apply to both DM3 and Springware boards, some time parameters may not be supported on DM3 boards. Also, ASCII characters are used to represent signal bit codes in the data structures. For example, "11xx" represents signal bits 11xx (where x represents "don't care"). All time parameters have units in millisecond with a resolution of 4 milliseconds.

The following define for the size of the CAS signal bits string is common to all three structures following:

```
#define     GCVAL_CAS_CODE_SIZE          0x5     /* The size of CAS Signal code in string */
```

CAS Transition Signal

```
/* Data structure for CAS Transition signal */
typedef struct {
    char            PreTransCode[GCVAL_CAS_CODE_SIZE];   /* ABCD pre-transition code */
    char            PostTransCode[GCVAL_CAS_CODE_SIZE];  /* ABCD post-transition code */
    unsigned short  PreTransInterval;                    /* The minimum time for the duration
                                                            of the pre-transition (in msec)*/
    unsigned short  PostTransInterval;                   /* The minimum time for the duration
                                                            of the post-transition (in msec)*/
    unsigned short  PreTransIntervalNom;                 /* The nominal time for the duration
                                                            of the pre-transition (in msec).
                                                            Ignored in DM3: always 0 */
    unsigned short  PostTransIntervalNom;                /* The nominal time for the duration
                                                            of the post-transition (in msec).
                                                            Ignored in DM3: always 0 */
} GC_CASPROT_TRANS;
```

CAS Pulse Signal

```
/* Data structure of CAS Pulse signal */
typedef struct {
    char            OffPulseCode[GCVAL_CAS_CODE_SIZE];    /* ABCD pulse off code */
    char            OnPulseCode[GCVAL_CAS_CODE_SIZE];     /* ABCD pulse on code */
    unsigned short  PrePulseInterval;                     /* The minimum time for the duration
                                                             of the pre-pulse (in msec) */
    unsigned short  PostPulseInterval;                    /* The minimum time for the duration
                                                             of the post-pulse (in msec) */
    unsigned short  PrePulseIntervalNom;                  /* The nominal time for the duration
                                                     of the pre-pulse. Ignored in DM3: always 0 */
    unsigned short  PostPulseIntervalNom;                 /* The nominal time for the duration
                                           of the post-pulse (in msec). Ignored in DM3: always 0 */
    unsigned short  PulseIntervalMin;                     /* The minimum time for the duration
                                                             of the pulse interval (in msec) */
    unsigned short  PulseIntervalNom;                     /* The nominal time for the duration
                                                             of the pulse interval (in msec) */
    unsigned short  PulseIntervalMax;                     /* The maximum time for the duration
                                                             of the pulse interval (in msec) */
} GC_CASPROT_PULSE;
```

CAS Train Signal

```
/* Data structure of CAS Train signal */
typedef struct {
    char            OffPulseCode[GCVAL_CAS_CODE_SIZE];    /* ABCD pulse off code */
    char            OnPulseCode[GCVAL_CAS_CODE_SIZE];     /* ABCD pulse on code */
    unsigned short  PreTrainInterval;                     /* The minimum time for the duration
                                                             of the pre-train (in msec) */
    unsigned short  PostTrainInterval;                    /* The minimum time for the duration
                                                             of the post-train (in msec) */
    unsigned short  PreTrainIntervalNom;                  /* The nominal time for the duration
                                                      of the pre-train. Ignored in DM3: always 0 */
    unsigned short  PostTrainIntervalNom;                 /* The nominal time for the duration
                                           of the post-train (in msec). Ignored in DM3: always 0 */
    unsigned short  PulseIntervalMin;                     /* The minimum time for the duration
                                                             of the pulse interval (in msec)*/
    unsigned short  PulseIntervalNom;                     /* The nominal time for the duration
                                                             of the pulse interval (in msec)*/
    unsigned short  PulseIntervalMax;                     /* The maximum time for the duration
                                                             of the pulse interval (in msec)*/
    unsigned short  InterPulseIntervalMin;                /* The minimum time for the duration
                                                             of inter-pulse interval (in msec)*/
    unsigned short  InterPulseIntervalNom;               /* The nominal time for the duration
                                                            of inter-pulse interval (in msec)*/
    unsigned short  InterPulseIntervalMax;               /* The maximum time for the duration
                                                           of inter-pulse interval (in msec) */
} GC_CASPROT_TRAIN;
```

CAS Signal Type Defines

The following new value types for CAS signal parameter are defined in the *gccfgparm.h* file to represent the CAS Transition, CAS Pulse, and CAS Train types, respectively. These defines are used by the **gc_QueryConfigData( )** for the value type of CAS signal.

```
    GC_VALUE_CAS_TRANS    =     0x10,    /* CAS Transition data struture ==> GC_CASPROT_TRANS */
    GC_VALUE_CAS_PULSE    =     0x11,    /* CAS Pulse data struture ==> GC_CASPROT_PULSE */
    GC_VALUE_CAS_TRAIN    =     0x12,    /* CAS Train data struture ==> GC_CASPROT_TRAIN */
```

Other value types (for example, integer, string, long, etc.) have already been defined in the *gccfgparm.h* file.

New Set IDs and Parm IDs

This feature uses the following new Set IDs and Parm IDs:

Set ID	Parm IDs
CCSET_LINE_CONFIG	CCPARM_LINE_TYPE CCPARM_CODING_TYPE
GCSET_PROTOCOL	GCPARM_PROTOCOL_ID GCPARM_PROTOCOL_NAME
PRSET_CAS_SIGNAL (defined in *dm3cc_parm.h*)	The parm ID is dynamically generated.
PRSET_TSC_VARIABLE (defined in *dm3cc_parm.h*)	The parm ID is dynamically generated.

1.11.3 Restrictions and Limitations

The following restrictions and limitations apply:

- This feature supports the redefinition of CAS signals and the setting of CDP variable values for a specific protocol variant, which will affect all channels running that protocol variant. It does not, however, support the getting or setting of protocol parameters on an individual channel basis. Getting and setting CAS signal definitions or CDP variable values is only supported for PDK protocols.
- Prior to changing parameters of a protocol, all channels running the protocol should be in the Idle state (that is, there should be no call activity on the channels). Once the parameter value change is complete, it is recommended to reset the channels running the affected protocol.
- At lease one time slot has to remain open while setting or retrieving CAS signal definitions or CDP variable values.
- It is recommended to receive the unblocked/blocked event of the first time slot opened prior to setting or getting any CAS signal definition or CDP parameter value.

- Using this feature to set/get multiple CAS signal definitions in a single GC_PARM_BLK via the **gc_SetConfigData( )** and **gc_GetConfigData( )** functions or mixing CAS signal definitions with other parameters is not supported. Only one CAS signal definition (and no other parameters) can be included in any one function call.
- The API for this feature can be used only after the board firmware has been downloaded.
- Configuration files are not updated with changes made using this API for this feature. The API does not save or store the changes made and if the firmware is re-downloaded, all information configured using this API will be lost. It is the API user's responsibility to save or store the changed configuration information and reset via the API in the event of a re-download.
- This feature supports the redefinition of CAS Transition, CAS Pulse and CAS Train signals only. In addition, this feature does not support the changing of the CAS signal type during redefinition. For example, the CAS_WINKRCV signal type cannot be changed from a CAS Pulse to a CAS Transition.
- Error checking and ensuring the validity of parameters passed through this API is the responsibility of the API user.
- The list of parameters that need to be modified must be managed at run time. Parameter updates are sent to the firmware one at a time as opposed to the parallel procedures used to set parameters at firmware download time. It is recommended to keep the list of parameters that need modification to a minimum.
- This feature supports the setting and retrieval of multiple CDP variable values in a single API call, but it does not support the mixing of CDP variables with other parameters when setting or retrieving values.
- To set the values of the CDP_IN_ANI_Enabled and CDP_OUT_ANI_Enabled in the *pdk_us_mf_io.cdp* file, the user is required to remove feature_ANI from the SYS_FEATURES section of the CDP file. Similarly, to set the values of the CDP_IN_DNIS_Enabled and CDP_OUT_DNIS_Enabled, the user is required to remove feature_DNIS from the SYS_FEATURES section.
- Some parameters may require a reset of the entire stack (for example, using a **gc_ResetLineDev( )**). Such requirements are parameter-specific and will be outlined in the documentation.

1.11.4 Documentation

The online bookshelf provided with the system release contains information about all system release features including features for application development, configuration, administration, and diagnostics.

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

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to E1 and T1 technology, see:

- *Global Call E1/T1 CAS/R2 Technology Guide*

For features specific to ISDN technology, see:

- *Global Call ISDN Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the manuals above do not contain information relating to this feature.

1.12 Support for MIME on Subscribe and Notify Messages

The capability to attach a MIME message body has been added to the Subscribe, Subscribed Notify, and Unsubscribed Notify messages and their 200 OK responses. The MIME message body can either be single-part or multipart MIME.

The steps necessary for adding MIME information follow the same process described in Section 4.10 of the Global Call IP for Host Media Processing Technology Guide.

Information on how to send and receive Subscribe/Notify messages and responses is provided in Section 4.15 of the Global Call IP for Host Media Processing Technology Guide.

1.13 Support for V.17 PCM Fax

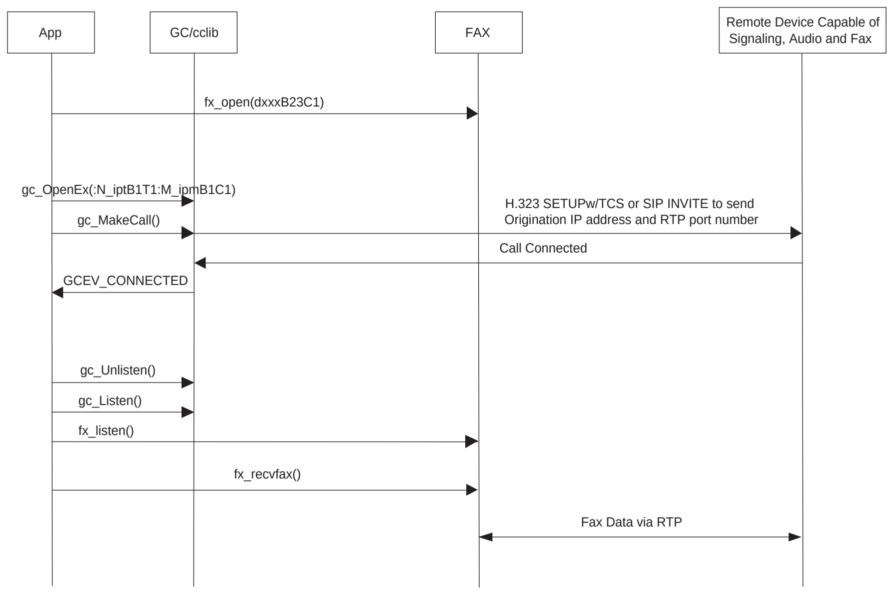
The HMP system is capable of originating and receiving a fax using a V.17 soft modem. This facility can be used with Global Call routing the V.17 PCM data over a G.711 coder in a “fax pass-through” mode by connecting the fax (dxxx) device directly to the media (ipm) device via a PCM connection. See Section 1.2.1 and Section 1.2.2.

Alternatively, the V.17 PCM data can be routed over a PSTN connection by connecting the fax (dxxx) device directly to the DTI front end device (refer to the *Fax Software Reference for Linux and Windows*).

1.13.1 Sending G.711 Fax in an Established Audio Session

In this scenario, the user application uses the Global Call API to open a Media device and make a voice call. A Fax device is then opened and the application connects the Fax device to the voice session. See Figure 1.

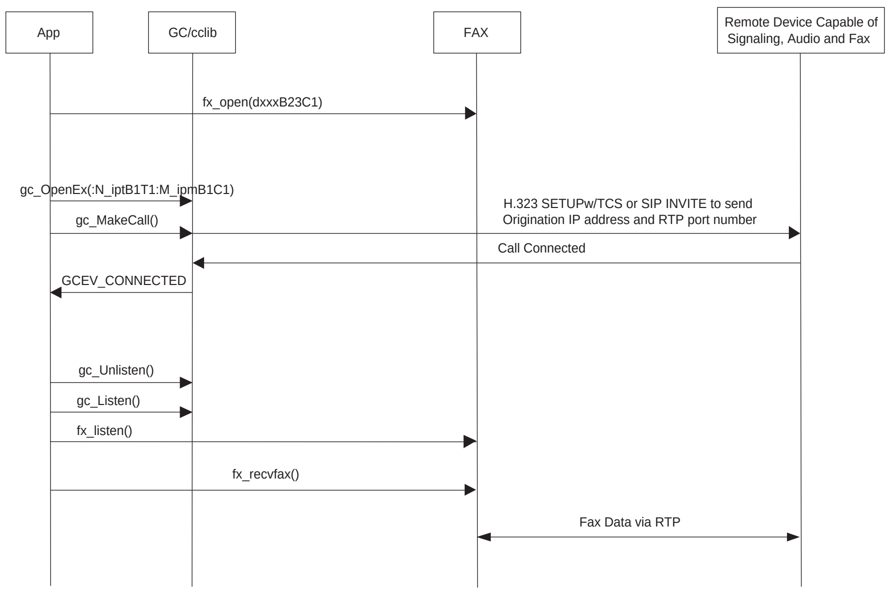
Figure 1. Sending G.711 Fax in an Established Audio Session


App
GC/cclib
FAX
Remote Device Capable of Signaling, Audio and Fax
fx_open(dxxxB23C1)
gc_OpenEx(:N_iptB1T1:M_ipmB1C1)
gc_MakeCall()
H.323 SETUPw/TCS or SIP INVITE to send Origination IP address and RTP port number
Call Connected
GCEV_CONNECTED
gc_Unlisten()
gc_Listen()
fx_listen()
fx_recvfax()
Fax Data via RTP

1.13.2 Receiving G.711 Fax in an Established Audio Session

In this scenario, the user application uses the Global Call API to open a Media device and establishes an audio session with the remote device. To prepare to receive a fax, the application must also open a Fax device. See Figure 2.

Figure 2. Receiving G.711 Fax in an Established Audio Session


App
GC/cclib
FAX
Remote Device Capable of Signaling, Audio and Fax
fx_open(dxxxB23C1)
gc_OpenEx(:N_iptB1T1:M_ipmB1C1)
gc_MakeCall()
H.323 SETUPw/TCS or SIP INVITE to send Origination IP address and RTP port number
Call Connected
GCEV_CONNECTED
gc_Unlisten()
gc_Listen()
fx_listen()
fx_recvfax()
Fax Data via RTP

For additional API function information, refer to the *Global Call API for Linux and Windows Operating Systems Library Reference* and the *Fax Software Reference for Linux and Windows*.

1.14 Mixed ISDN, CAS and R2MF Protocols

HMP supports the mixing of ISDN protocols with CAS or R2MF protocols on the trunks of a DNI601TEPHMP board, and the mixing of ISDN protocols and CAS protocols on the trunks of a DNI1200TEPHMP board.

For additional information, refer to the *Dialogic® Digital Network Interface Boards Configuration Guide*.

Release Issues 2

This chapter includes the following topics that relate to release issues:

2.1 Issues

The following table lists issues that can affect the software supported in Dialogic® Host Media Processing Software 2.0WIN. 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:

- Resolved – An issue that was resolved (usually either fixed or documented) in this release.
- Known (permanent) – A known issue or limitation that will not be fixed in the future.
- Known – A minor issue that affects the Dialogic® Host Media Processing Software 2.0WIN. This category includes interoperability issues and compatibility issues. Known issues are still open but may or may not be fixed in the future.

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.).

PTR No.
: Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the defect number is used to track the issue.

SU No.
: For defects that were resolved in a Service Update, indicates the Service Update number. 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, typically one of the following:

- a system-level component; for example, Dialogic® HMP 2.0WIN
- a software product; for example, the Global Call Library
- a software component; for example, Continuous Speech Processing

Description

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

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00078606	--	184	Fax	When the fax channel begins to receive, the application randomly stops the channel, and all of the fax channels hang up.
Resolved	IPY00044978	--	183	Fax	The **fx_rcvfax( )** function fails with a -1 error.
Resolved	IPY00044782	--	182	Dialogic® HMP software	A race condition was discovered while decrementing a reference count.
Resolved	IPY00044700	--	182	Dialogic® HMP software	The Radvision asn.1 library crashes.
Resolved	IPY00044620	--	182	Fax	The Fax resource remains stuck in a STOP state.
Resolved	IPY00044488	--	182	Global Call IP (SIP)	When the CGPN and CDPN optional IEs are not present in an incoming setup message, the H323 stack does not put anything in.
Resolved	IPY00042011	--	181	Configuration	The Dialogic® Configuration Manager (DCM) and the DCM system tray has inconsistent system status.
Resolved	IPY00044437	--	181	Fax	Fax channels hang intermittently.
Resolved	IPY00044273	--	181	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	--	181	Global Call IP (SIP)	Manual transfer ends with a disconnected event.
Resolved	IPY00043261	--	181	Global Call IP (SIP)	In 1PCC, a HOLD reINVITE using the **gc_AcceptModifyCall( )** function fails for low bit rate codecs.
Resolved	IPY00042653	--	181	IP Media	DTMF packets are missing.
Resolved	IPY00040101	--	180	CFSP	Windows® 2003 Server crashes due to an APICDRV.sys issue.
Resolved	IPY00042993	--	180	CSP	The **ec_getblkinfo( )** returns incorrect info when the stream buffer size is 4096 bytes or larger.
Resolved	IPY00041047	--	176	Fax	Phase D Reply information in not returned in a timely manner when performing a Receive Fax operation.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00042329	--	176	Global Call IP (SIP)	When programmed in 1PCC GCCAP_DONTCARE, the SIP G.729AB ingress call sets an incorrect RX media codec.
Resolved	IPY00042186	--	176	IP Media	Changing the payload type to 101 in an RTP stream maxes out the jitter buffer, and results in unacceptable latency.
Resolved	IPY00041686	--	175	Fax	In manual modify mode, H.323 functionality breaks with regard to transitioning from audio to fax.
Resolved	IPY00041789	--	175	Global Call IP (SIP)	There is no checking of UPDATE in the Allow Header of the Ingress INVITE/ 200 OK.
Resolved	IPY00041876	--	175	IP Media	H.323 transfer fails with the RTF log ERR1 message, EVENT_CONNECT_SIGNALING failed.
Resolved	IPY00041405	--	175	IP Media	When the **gc_Listen( )** function is executes in async mode, an IMPEV_LISTEN event is received instead of a GCEV_LISTEN event.
Resolved	IPY00041112	--	173	Conferencing	The application and drivers do not respond when attempting to change the attributes on a party device that is already in a conference.
Resolved	IPY00041063	--	173	Conferencing (DCB)	The **dcb_unmonconf( )** function unmonitors the wrong conferee (party).
Resolved	IPY00040029	--	173	Conferencing (DCB)	A memory leak occurs when a conference is established and then deleted quickly and frequently.
Resolved	IPY00041268	--	173	Configuration	Incorrect mutex causes memory corruption.
Resolved	IPY00039536	--	173	Configuration	D-Channel tracing with Dialogic® HMP DNI boards time stamps are not aligned with the system time.
Resolved	IPY00041583	--	173	Dialogic® HMP	Dialogic® HMP sends the bye message, IPEC_SIPReasonStatusBYE = 5800, upon making a call after the remote Nortel CS1000 sends an additional media type.
Resolved	IPY00041847	--	173	Fax	A BSOD occurs on a system running heavy amounts of faxing activity.
Resolved	IPY00041570	--	173	Fax	The Fax resource causes the driver to lockup.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00038208	--	173	Global Call	When an incoming ISDN PROGRESS message contains a cause IE, the Call Control library does not forward it along with the GCEV_PROGRESSING event, or clear it. Instead this cause value is associated with the GCEV_DISCONNECTED event.
Resolved	IPY00041815	--	173	Global Call IP (SIP)	Call control failures occur when an early media SIP reINVITE parameter is added to the media load CONFIG file.
Resolved	IPY00040743	--	173	Global Call IP (SIP)	There is excessive logging generated by SIP INFO messages.
Resolved	IPY00039823	--	173	Global Call IP (SIP)	When modifying the "ReferTo" header field, the Global Call API only adds content to the end of the SIP header instead of replacing the contents of the pre-existing header field. This results in two "ReferTo" header fields sent in the outgoing SIP REFER message when invoking the call transfer.
Resolved	IPY00041296	--	173	H323 Call control	In the LD state MediaDrop, the event EVENT_CHANNELS_OPENED is handled only when the call is in the connected state. This event should be handled in alerting and proceeding states too.
Resolved	IPY00041280	--	173	H323 Signal	Codec validation does not occur when the remote side offers fax only codec for an H323 slowstart call.
Resolved	IPY00041773	--	173	IP Media	There is an exception in RTCP handling when the number of received SSRCs is large.
Resolved	IPY00042016	--	173	Voice	**ATDX_BUFDIGS( )** returns an incorrect value.
Resolved	IPY00041254		173	Voice	Two inits of the same CSP Channel result when two **dx_open( )** calls are made to the same channel.
Resolved	IPY00040440	--	160	Global Call IP (SIP)	A fall back occurs when DNS resolution fails while making an SIP HMP call.
Resolved	IPY00040803	--	160	IPML	If **ipm_StartMedia( )** is called with an invalid IP address (0.0.0.0), all subsequent calls to the **ipm_StartMedia( )** function return an "Invalid Parameter" error.
Resolved	IPY00037476	--	160	PSTN	The NFAS standby or backup D-channel fails to work when configured for 4ESS.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00039964	--	160	Voice	The Recorder component has DTMF back suppression parameters in millisecond units, while the Encoder uses sample units.
Resolved	IPY00040031	--	159	Conferencing (DCB)	The **dcb_SetParm( )** function disables Active Talker Detection on only the first conference.
Resolved	IPY00040364	--	159	Configuration	Dialogic® Services do not start when you add "SetParm=0x401b,1" into the configuration file for RFC2833 enhancements.
Resolved	IPY00040082	--	159	Configuration	When stopping Dialogic® Services via DCM, the control panel indicates a stop a few seconds before DCM completes the task.
Resolved	IPY00039701	--	159	Configuration	Toggling the Dialogic® Service start up state between Manual and Semi-Automatic mode also toggles the board download state provided by **NCM_GetSystemState( )**.
Resolved	IPY00039410	--	159	Fax	Fax channels do not respond to the **fx_stopch( )** function.
Resolved	IPY00039677	--	159	Global Call	The **gc_ReleaseCall( )** function does not receive a termination event after the application attempts to handle a GCEV_TASKFAIL.
Resolved	IPY00040038	--	159	Global Call	Two TASKFAILS result when a BUSY is returned during a blind transfer in response to switch NOTIFY messages.
Resolved	IPY00040637	--	159	Global Call IP	A Quality of Service (QOS) error is generated when using Interactive Voice Response (IVR) servers and Dialogic® HMP software.
Resolved	IPY00040033	--	159	Global Call IP (SIP)	For an incoming reINVITE, a “488” response is not sent automatically when reINVITE support is disabled.
Resolved	IPY00039707	--	159	Global Call IP (SIP)	A "glare" condition resulting in a disconnected call occurs during an automatic SIP reINVITE when media switches from audio to fax.
Resolved	IPY00037336	--	158	Configuration	The dm3nk INFO and ADVISORY messages posted in the event log cause confusion.
Resolved	IPY00039855	--	158	Fax	Implement better handling of the invalid Data Modulation Bits from a DIS frame.
Resolved	IPY00039505	--	158	Fax	A request to switch to T.38 fax is rejected, but a gcev_taskfail event is not received.
Resolved	IPY00039639	--	158	Global Call	The **gc_SetAlarmParm( )** function causes an assert in *msvcrt.dll*.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00039451	--	158	Global Call	The **gc_AcceptModifyCall( )** function fails to return a termination event.
Resolved	IPY00039564	--	158	Global Call IP (SIP)	A GCEV_EXTENSION event (IPSET_SWITCH_CODEC, IPPARM_READY) is not received for T.38 call.
Resolved	IPY00039401	--	158	Global Call IP (SIP)	The "Record-Route" field of the SIP header message gets incorrectly reported as the "Route" field in an incoming SIP message when using the Global Call API.
Resolved	IPY00039333	--	158	Global Call IP (SIP)	The SIP INVITE fast-start codec G.729A should default to annex B.
Resolved	IPY00039315	--	158	Global Call IP (SIP)	An Access Violation (0xC0000005) occurs in *LIBSIPSIGAL.DLL* when receiving a SIP REFER message containing a header field parameter with content greater than 255 bytes.
Resolved	IPY00039965	--	158	GlobalCall (IP)	An outbound IP call fails with "IPEC_SIPReasonStatus503ServiceUnavailable" when the hostname is passed for the destination address in the dialstring.
Resolved	IPY00039409	--	158	IPML	An incorrect eQoSType, QOSTYPE_ROUNDTRIPLATENCY, is specified for QoSRTCPTimeout and QoSRTPTimeout.
Resolved	IPY00039847	--	158	Signaling (H.323)	A RequestMode is not triggered upon CED detection in AUTO mode.
Resolved	IPY00039675	--	156	Fax	A system crash occurs in the *ssp.mlm.sys* file.
Resolved	IPY00039649	--	156	Fax	The application fails to perform an **fx_listen( )** on V.17 fax sessions.
Resolved	IPY00039249	--	156	Global Call	When the application issues a **gc_WaitCall( )** after an inbound call, **gc_AcceptCall( )** fails, and a GCEV_OFFERED event is received.
Resolved	IPY00039179	--	156	Global Call	When the SYNC function **gc_ReleaseCall( )** is called during a glare scenario, an unexpected GCEV_RELEASECALL event is returned.
Resolved	IPY00039852	--	156	Installation	A slipstream install error occurs.
Resolved	IPY00039321	--	150	Fax	The preamble DIS frame does not respond during a fax.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00039206	--	150	Fax	The **fx_sendfax( )** function fails with a TFX_FAXERROR response.
Resolved	IPY00039225	--	150	Global Call IP (SIP)	There is an incorrect wrong codec in the SIP 200OK SDP. It should be G729a because the INVITE contained G729a, rather than G729ab.
Resolved	IPY00039036	--	150	Global Call IP (SIP)	The application returns a "486 Busy Here" response when closing and reopening a device in 3PCC mode.
Resolved	IPY00039033	--	148	Fax	The **fx_sendfax( )** function hangs in PHASE D.
Resolved	IPY00038992	--	148	Global Call IP (SIP)	The application is missing a termination event after executing the **gc_InvokeXfer( )** function followed by a disconnect from the receiving end.
Resolved	IPY00038849	--	148	SRL	The **sr_enbhdlr( )** function fails to re-allocate memory for handler array after an attempt to open all channels in a multiple board configuration.
Resolved	IPY00038856	--	146	Fax	A Fax resource error occurs after applying a patch for IPY00038205.
Resolved	IPY00037654	--	146	Fax	After a successful call to **fx_rcvfax( )** in T.30 mode, the entire received TIFF file gets the wrong image contents when ECM is used and retransmission occurs.
Resolved	IPY00038848	--	146	Global Call	When **gc_DropCall( )** returns a cause code of 5801 from the application, an invalid response code appears in the RTF log.
Resolved	IPY00038732	--	146	Global Call IP	The transmit codec transmits at 6.3 kbps despite being programmed at G.723.1 5.3 kbps.
Resolved	IPY00038663	--	146	Global Call IP	A Quality of Service (QOS) error is generated when using Interactive Voice Response (IVR) servers and HMP software.
Resolved	IPY00038868	--	146	Global Call IP (SIP)	A GCEV_CONNECTED event is not returned from outgoing calls using SIP.
Resolved	IPY00038827	--	146	Global Call IP (SIP)	The **gc_AnswerCall( )** function does not receive a response when a SIP CANCEL is received after a SIP INVITE.
Resolved	IPY00038908	--	146	SNMP	SNMP service crashes periodically.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00038708	--	146	SRL	An access violation occurs during **sr_freedata( )** causing the application to stop.
Resolved	IPY00038218	--	145	Device Management	An exception occurs in the Device Management library when **dev_SetError( )** keeps data in local thread storage.
Resolved	IPY00038288	--	145	Fax	An attempt is made to free a NULL pointer after a memory allocation failure.
Resolved	IPY00038580	--	145	Global Call IP (SIP)	An access violation occurs in *libsipsigal.dll*.
Resolved	IPY00038572	--	145	Global Call IP (SIP)	Unable to determine the IPPARM_MGSTYPE value for an incoming SIP message.
Resolved	IPY00038474	--	145	Voice	The voice resource gets stuck in a busy state when a TDX_CALLP event is not returned from the **dx_dial( )** function.
Resolved	IPY00038313	--	144	Fax	The Fax application does not return any event when the channel hangs after calling the **fx_sendfax( )** function.
Resolved	IPY00038513	--	144	HMP	UIO offset does not account for data already played from a previous buffer.
Resolved	IPY00038549	--	144	Voice	Ports are stuck in a CS_STOPD state when a caller hangs up during playback.
Resolved	IPY00038475	--	144	Voice	The **dx_dial( )** function does not always a return a TDX_CALLP event when stopped with the **dx_stopch( )** function.
Resolved	IPY00038217	--	143	Conferencing (DCB)	The **dcb_setcde( )** function fails with an "invalid board" error.
Resolved	IPY00038470	--	143	Global Call IP	A GCEV_CONNECTED event is missing from the RTF log.
Resolved	IPY00038365	--	143	Global Call IP (SIP)	Egress SIP calls work briefly, then omit SDP (G711A codecs) in egress INVITE messages.
Resolved	IPY00038343	--	143	Global Call IP (SIP)	When creating registrations, the customer runs into a limit and gets an error unless he unregisters a previous user.
Resolved	IPY00038342	--	143	Global Call IP (SIP)	Simultaneous disconnects cause an error at RTL after a number of calls.
Resolved	IPY00037358	--	143	HMP	The receipt of RFC 2833 digits after the G.711 Comfort Noise packet causes HMP to double detect the first RFC 2833 digit incorrectly.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00037603	--	143	Installation	The SNMP component gets installed despite not choosing it as an option during System Release installation.
Resolved	IPY00037756	--	140	Fax	HMP fails to deliver a termination event for "send fax" when the **fx_stopch( )** function is called randomly
Resolved	IPY00038150	--	140	Global Call IP	A simultaneous re-INVITE from both sides of call causes a TASKFAIL for the **gc_ReqModifyCall( )** function.
Resolved	IPY00038240	--	140	Global Call IP (SIP)	A thread in the API is asserting when an inbound re-INVITE is received before GCEV_REQ_MODIFY_CALL.
Resolved	IPY00038060	--	140	Global Call IP (SIP)	A rvSdpStringGetData crash occurs when there are no media attributes containing "rtpmap" in a SIP INVITE.
Resolved	IPY00033127	--	140	Installation	Application is unable to get an unblocked event while using the PDK_US_LS_FXS_IO protocol.
Resolved	IPY00037699	--	139	Global Call IP (H.323)	The **gc_Stop( )** function fails to return control to the application when called after the LAN cable is disconnected.
Resolved	IPY00036779	--	139	Global Call IP (SIP)	Assertion faults are received when "DON'T CARE" is used with the G.726 coder.
Resolved	IPY00037776	--	139	PSTN Call Control	When using a Quad Span thin blade, ISDN firmware running NI2 quickly disconnects a normal outbound call that has progressed through the ALERTING/PROCEEDING/PROGRESSING states.
Resolved	IPY00037422	--	137	Global Call	When using DNI boards running ISDN on HMP2.0 SU127 and encountering frame slips on the line, calls to the **gc_SetConfigData( )** function to change line properties causes the application to either hang completely or return after an extremely long time.
Resolved	IPY00037825	--	137	Global Call IP (SIP)	The **gc_InvokeXfer( )** function fails.
Resolved	IPY00037778	--	137	Global Call IP (SIP)	The Session Description Protocol (SDP) is missing from SIP re-INVITE after calling the **gc_ReqModifyCall( )** function.
Resolved	IPY00037737	--	137	Global Call IP (SIP)	The **gc_MakeCall( )** function returns a GCEV_TASKFAIL event when frames per packet are set to 20.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00037733	--	137	Host Interface	A switch is responding to outbound calls with a DISCONNECT and cause code 0x19 (== 25). This cause code is not defined in the HMP header file.
Resolved	IPY00037777	--	137	Voice	The callback function fails after calling the **dx_stopch( )** function to stop **dx_playiottdata( )**.
Resolved	IPY00037655	--	136	Fax	While in T.30 mode, the **fx_rcvfax( )** function does not terminate when there is a busy tone.
Resolved	IPY00037578	--	136	Global Call IP	Echo cancellation fails after the first call on a device.
Resolved	IPY00037613	--	136	Global Call IP (SIP)	The **gc_InvokeXfer( )** function does not return a termination event.
Resolved	IPY00037561	--	136	Host Interface	The DM3 timeslot crashed.
Resolved	IPY00037633	--	136	PSTN Call Control	Call transfer fails when using protocol pdk_sw_e1_ssls_io.
Resolved	IPY00037541	--	136	SIP Signal	Upon receipt of the GCEV_REQ_MODIFY_CALL event, the RTP port-address info in the event data stays unchanged until the **gc_AcceptModifyCall( )** function is called and the GCEV_ACCEPT_MODIFY_CALL event is received.
Resolved	IPY00036658	--	136	SRL	The **gc_GetMetaEvent( )** function is failing with an Internal (SRL) failure.
Resolved	IPY00037530	--	136	Voice	Recording to memory using the **dx_reciottdata( )** function causes distortion at end of file.
Resolved	IPY00034857	--	134	DNI firmware	A GCCT_UNKNOWN event is returned for media detection if the media detection occurs before the out-of-band CONNECT message is received.
Resolved	IPY00037172	--	133	Global Call IP	The application returns a taskfail with IPERR_INVALID_PHONE_NUMBER as the reason.
Resolved	IPY00037333	--	133	Global Call IP (SIP)	There is no audio when calling from a digital phone through Alcatel PBX to HMP 1.3.
Resolved	IPY00036223	--	131	Fax	The HMP Fax channel hangs when calling **fx_stopch()** after receiving a Fax Phase B event.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00036950	--	131	Global Call IP	Incoming call lines get stuck in an invalid state after receiving a GCEV_TASKFAIL event.
Resolved	IPY00036618	--	131	Global Call IP (SIP)	HMP rejects register messages after several days of taking calls.
Resolved	IPY00034365	--	131	Voice	Enabling voice in the run-time tracing facility (RTF) config file causes a non-recoverable fatal error.
Resolved	IPY00036021	--	129	Configuration	HMP 2.0 cannot open the board device on the thin blade more than once.
Resolved	IPY00032866	--	129	Voice	If a user attempts to play forever or until end of file (EOF) is reached (specifying io_length = -1), and User Input Output (UIO) plays on DM3/HMP, there is a hard upper limit on the number of bytes that can be played, which is approximately equal to 2.147GB (~2 to the 31 bytes).
Resolved	IPY00034254	--	127	Global Call	The **gc_SetConfigData( )** function hangs when changing line encoding/framing at runtime.
Resolved	IPY00035675	--	127	HMP	Server locks up (freezes) during CAS Bulk Call test under HMP2.0 SU using QUAD span.
Resolved	IPY00036346	--	127	IP Media Server	One of the ec_ functions causes system crash in IPMediaServer on HMP 2.0.
Resolved	IPY00036301	--	127	IP Media Session Control (RTP)	While switching between two RTP streams, a few seconds of silence duration is created in recorded file.
Resolved	IPY00036291	--	127	Licensing	HMP 2.0 regenerates config/pcd/fcd files upon reboot and erases all changes that were previously made.
Resolved	IPY00033734	--	123	Fax	Stuck channels are not returning events for **fx_stopch()** and **fx_sendfax().**
Resolved	IPY00033723	--	123	Fax	In HMP, when the application calls **fx_stopch()** on a channel that is sending or receiving fax, the channel will become unusable and cannot receive any event any more until restarting the HMP.
Resolved	IPY00035875	--	123	Global Call IP	Global Call IP applications that were recompiled using Service Updates 116, 119, 120 or 122 need to be recompiled again with Service Update 123 or later.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00035631	--	122	Configuration	There is a memory leak in **NCM_GetFamilyDeviceByAUID()** function.
Resolved	IPY00035822	--	122	Global Call IP	HMP 2.0 & 1.3 do not respond to SIP 407 Proxy Authentication Required messages.
Resolved	IPY00033573	--	122	Host	In certain scenarios, Orbacus services fail on startup.
Resolved	IPY00034525	--	122	SNMP	C:\dlgagent.log file increasing in size as snmp.exe writes same entry in this file every 10 seconds: INVALID_HANDLE_VALUE - 2.
Resolved	IPY00035644	--	120	Fax	Corrected number of EOLs provided during T.38 operation.
Resolved	IPY00034416	--	120	Global Call IP	Assistance with SIP re-INVITE for existing call does not change RTP port.
Resolved	IPY00034840	--	120	IP Media Session Control (RTP)	**ipm_ResetQoSAlarmStatus()** fails to reset QoS alarms allowing the same device to not receive the same alarm again
Resolved	IPY00035622	--	120	Voice	There are problems playing a file from a remote source if the source is disconnected and exception is not handled correctly.
Resolved	IPY00034311	--	119	Configuration	The CAS T1 config and FCD files default to D4/B8ZS instead of D4/AMI for all generated T1 CAS protocol files in HMP2.0SU line for the DNI boards.
Resolved	IPY00034309	--	119	Configuration	When downloading services on HMP2.0 with a DNI1200TEPHMP board installed, sometimes an event log warning is logged by dm3nk, which is not easy to decipher.
Resolved	IPY00035028	--	119	Global Call IP	The Mitel 3300 sends an empty realm string (realm="") and HMP does not respond with any username/password.
Resolved	IPY00034787	--	119	IP Media Session Control (RTP)	Application no longer receives events after recovering a long period of LAN disconnect. System must be rebooted to recover.
Resolved	IPY00034499	--	116	Conferencing	When running conferencing activity under high load on dual core systems, there is a potential for **dcb_delconf()** to hang and never return.
Resolved	IPY00034440	--	116	Fax	Synchronization object timeout errors with **ipm_StartMedia( )**.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00034316	--	116	Fax	Send fax attempts fail with Transmit Communication Error (190) on HMP 1.3 after EOM fax protocol message is sent to receiving side running Hylafax Application (Software based fax receiver).
Resolved	IPY00034403	--	116	HMP	ATDV_SUBDEV returns incorrect value (number of devices) for ISDN spans under HMP2.0 using the mixed protocol (ISDN & CAS) DUS-100829682.
Resolved	IPY00034318	--	116	HMP	Issuing dx_stopch with the mode parameter set to EV_NOSTOP on idle channel does not return TDX_NOSTOP event.
Resolved	IPY00034314	--	116	HMP	Time display in fax header of AM or PM is wrong for certain time period.
Resolved	IPY00033740	--	116	HMP	V.17 fax call fails after a successful T.38 fax call on same fax resource.
Resolved	IPY00033598	--	116	HMP	Same issue that caused BSOD on HMP 1.1 SU37 servers will cause it on HMP 2.0.
Resolved	IPY00033552	--	116	HMP	APIC Driver issue during calibration for bus speed can cause choppy audio.
Resolved	IPY00033403	--	112	Global Call	The function **gc_SetConfigData()** fails to complete.
Resolved	IPY00034196	--	112	HMP	Depending on the timing of the previous call tear down procedure, the next **gc_MakeCall( )** attempted on CAS lines running pdk_us_mf_io on DNI1200TEPHMP boards on HMP 2.0 could yield no further events.
Resolved	IPY00033973	--	112	HMP	The PDK R2 protocol does not detect the idle pattern generated by the remote side from the seize state.
Resolved	IPY00033630	--	112	HMP	On DM3, ABCD bits go to BLOCKING (1101) pattern immediately after download, on DNI, ABCD bits first come up with SEIZED (0001) pattern, and then got to BLOCKING after about one minute.
Resolved	IPY00033495	--	112	HMP	HMP 2.0 SU97 intermittently fails to start service and sometimes requires a reboot before services can start again even if no configuration changes have been made.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Resolved	IPY00033474	--	112	HMP	If using HMP 2.0 on windows XP Embedded and downloading a PDK protocol for spans on a DNI1200TEPHMP board, services will start successfully, but under certain circumstances, PDK will not download properly because of a failure in an internal system call.
Resolved	IPY00032398	--	104	Voice	During Tranasaction Record scenarios, empty files were recorded.
Resolved	IPY00032239	36769	97	Global Call IP	There is a problem when using IPPARM_SIP_HDR to set call ID. Somehow it is not recognized by subsmanager in SIP signal.
Resolved	IPY00031798	36748	94	IP Media Server Demo	After changing the mainMenu.vox prompt to a duration of <= 0.55 seconds, the prompt takes several seconds before playing.
Resolved	IPY00031799	36730	94	Voice	Recording only starts after an 8 second delay. For example, a recording of 25 seconds only records the last 17 seconds. A recording of less than 8 seconds results in an empty file.
Known	IPY00006325	36540		Licensing	If the license file is located on a network drive, the licensing service may not start correctly. ***Workaround:*** Recommend that license file be stored on local system.
Known	IPY00006356	36412		Conferencing	If conferencing application is abnormally stopped during conferencing, HMP service may fail to start because of not being stopped correctly. ***Workaround:*** None
Known	IPY00006121	36098		Conferencing	The functions **ipm_getxmitslot()** and **dcb_addtoconf()** will timeout on 400 channels. ***Workaround:*** Use no more than 254 conferees per conference on a single conference.
Known (permanent)	--	--		Conferencing	The **dcb_setbrdparm()** function fails when attempting to set MSG_ALGORITHM to ALGO_LOUD and ATDV_ERRMSGP returns "Bad global parameter value". Do not set this parameter. By default, the algorithm uses the Loudest Talker.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known	IPY00006156	36002		CSP	When using CSP, the **ec_stopch()** function may hang in some streaming to board tests. ***Workaround:*** None
Known	IPY00006236	35116		DCM	Pop-up error messages occur when "Country" tab is selected in DCM. ***Workaround:*** Do not select Country tab. It is not supported or required.
Known	IPY00006281	36474		Demo	Call failures occur in the IP Gateway (Global Call) Demo when calls are made in the IP to PSTN direction. ***Workaround:*** Stagger the calls in the IP to PSTN direction.
Known	IPY00006280	36472		Demo	The IP Gateway (Global Call) Demo may fail when more than six calls are made in the PSTN to IP direction when there is no staggering between calls. ***Workaround:*** Stagger the calls in the PSTN to IP direction.
Known (permanent)	--	--		Demo	Demos are intended to be run using a maximum of 4 channels.
Known	IPY00006332	36534		Demos	The following demos may not compile if moved from the default directory location: • conferencing • ipmediaserver • sitest • speechprocessing ***Workaround:*** Leave these demo files in the default location following installation of the HMP software.
Known	IPY00005961	31271		Device Management	If an error occurs when executing a Device Management API library function, a value of -1 is returned. When SRL function **ATDV_LASTERR()** is called, it returns 0 to indicate no error. ***Workaround:*** This is not a problem if an error occurs within a subsystem (e.g., IP Media library) as **ATDV_LASTERR()** will return the correct error code.
Known	IPY00006055	36034		Diagnostics	Some features of the Listboards utility do not work. ***Workaround:*** None

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known	IPY00006295	35985		DNI Board	When the d-channel comes back up and channels try to make calls at the same time, only the first channel will correctly connect. ***Workaround:*** Allow a few seconds of delay before making calls.
Known	IPY00006252	35312		DNI Board	When the logical ID of the HMP board it set to a high number, the download may fail. ***Workaround:*** Only use logical IDs less than 255.
Known	IPY00028507	36610		DSI Board	DSI single board start/stop may not work consistently on some systems. ***Workaround:*** Do not use the single board start/stop feature.
Known	IPY00006375	36608		DSI Board	At startup, multiple DSI boards may be assigned the same logical ID in DCM. ***Workaround:*** Using DCM, modify the logical IDs of the DSI boards such that each board has a unique logical ID.
Known	IPY00006374	36607		DSI Board	Removal of a DSI board after installation may result in subsequent startup failures. ***Workaround:*** Reboot the system.
Known	IPY00006367 (IPY00011030, IPY00006366)	36591 (36590, 36589)		DSI Board	Starting a DSI board as the clock slave may occasionally result in poor voice quality. ***Workaround:*** Restart HMP service.
Known	IPY00006338	36546		DSI Board	Starting a previously stopped clock master, which takes over as the master (due to clock priority precedence) does not work properly. ***Workaround:*** Do not change the default clock priority (6) for DSI Boards. Additionally, it is recommended that a DNI board be selected as the clock master.
Known	IPY00006304	36175		DSI Board	The libMeaSiapi.dll will fail when **si_Open()** and **si_Close()** are called concurrently from multiple threads on the same station in the same process. This failure can occur in more than one context (UMBC callback and SI station thread). ***Workaround:*** Open all SI stations at the beginning of the application and close them before exiting the application.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known	IPY00006215	36611		DSI Board	System densities of 128 DSI ports are not supported. ***Workaround:*** Limit the total number of DSI ports to 48.
Known	IPY00006676	30626		Fax	The TO: field in the fax header for polled faxes is empty. ***Workaround:*** None
Known	IPY00006575	34131		Fax	If using Debug View when making T.38 fax calls with a Cisco AS5300, there may be an occasional "QT38[3]: ExecIFPInProcessHSHDLC: unexpected ind: 0" error message. ***Workaround:*** None is required. Ignore this message.
Known	IPY00006546	33789		Fax	If using Debug View when making T.38 fax calls, there may be an occasional "FC3_ReadFrame call FAILED" error message. ***Workaround:*** None required as there is no error. Ignore the message.
Known (permanent)	--	--		Fax	For fax applications, the header file *srllib.h* must appear before the header file *faxlib.h* in the #include directive.
Known (permanent)	--	--		Global Call IP	During a SIP call termination, the Global Call application may fail to receive a BYE message from a remote user agent due to excessive network errors. If this occurs on your network with your SIP application, the application should be designed to include detection of RTP timeout alarms. When an RTP timeout alarm is received and the resource has not received a GCEV_DISCONNECTED event, the resource should drop the call and proceed as if it had received the GCEV_DISCONNECTED event.
Known (permanent)	--	--		Global Call IP	G.711 µ-Law and A-Law IP Encoding/Decoding only support 64 Kbps.
Known (permanent)	--	--		Global Call IP	Global Call applications using T.38 should call **gc_SetUserInfo(IPPARM_T38_CONNECT)** after GCEV_OFFERED, but before the next GC function call, for T.38 only calls. Otherwise, the call will fail and the application will get a GCEV_TASKFAIL event.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known (permanent)	--	--		Global Call IP	Host applications should always clean up resources before exiting. If the application terminates irregularly (resources are not cleaned up as described below), the HMP devices should be restarted using DCM to stop and then start the devices. • If the application is using Global Call for call control, the application should terminate all calls by issuing **gc_DropCall()** followed by **gc_ReleaseCallEx()**, or use **gc_ResetLineDev()**. Any open devices should then be closed using **gc_Close()**. The Global Call Libraries should then be stopped using **gc_Stop()** to release reserved resources. • All other resource types should be stopped and closed using the appropriate **xx_stop()** and **xx_close()** API function. (For conferencing applications, **dcb_delconf()** should be issued on existing conferences before closing the DCB device handle).
Known (permanent)	--	--		Global Call IP	If a call is made to HMP using NetMeeting* from a machine that does not have a sound card, the coder negotiation will fail and the call will be disconnected before a GCEV_ANSWERED event is generated.
Known (permanent)	--	--		Global Call IP	If a call is made to HMP using NetMeeting, the "secure outgoing calls" option must not be selected.
Known (permanent)	--	--		Global Call IP	In SIP Digest Authentication, Global Call only responds to one WWW-Authenticate or Proxy-Authenticate header (if there are multiple) in a 401 or 407 response.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known (permanent)	--	--		Global Call IP	Network conditions may cause UDP packets to be lost, some of which may contain the SIP "100 Trying" or "180 Ringing" non-reliable response messages that correspond to the GCEV_PROCEEDING and GCEV_ALERTING events respectively. Because SIP does not require retransmission of 1xx response messages, these events will not be generated when a UDP packet containing the corresponding 1xx response is lost. Applications should be written to continue processing a call when GCEV_CONNECTED is received, regardless of whether GCEV_PROCEEDING or GCEV_ALERTING was received.
Known (permanent)	--	--		Global Call IP	When making 240 calls to the same destination in burst mode, the application may receive GCEV_DISCONNECTED with error IPEC_Q931TransportError. This occurs when the remote side is not able to keep up with the socket connections (WSAECONNREFUSED 10061). The application should drop and release the call.
Known (permanent)	--	--		Global Call IP	When running Global Call H.323 applications on HMP with Cisco 2600 Gatekeeper (IOS Version 12.3 (1a)), the Cisco 2600 sends packets to the wrong destination port after switching a voice call to T.38 Fax. This can be avoided by enabling tunneling on both sides.
Known (permanent)	IPY00036934	--		Global Call IP (SIP)	Dialogic® HMP starts RTP after the reception of ACK to 200OK. This may cause the ICMP error "Destination Unreachable" at the remote endpoint if the endpoint starts transmitting RTP immediately after receiving 200OK. This issue occurs intermittently and only with a limited set of endpoints.
Known (permanent	IPY00028239	32740		HMP	After several cycles of starting and stopping the System Service, or a single device (HMP), the System Service may hang if any Debug View client (DBMON or Debug View) is active.
Known (permanent)	IPY00010888	36193		HMP	Operating HMP with Black Ice* installed will result in degraded system performance (e.g. voice quality). Do not operate HMP with Black Ice* on the system.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known (permanent)	IPY00010577	36051		HMP	Some machines may briefly freeze during download before completion.
Known	IPY00006336	36435		HMP	If the autostart option is changed while HMP service is running, a failure may result when HMP service is stopped. ***Workaround:*** Do not change the autostart option while HMP service is running.
Known	IPY00006256	35630		HMP	Stopping the board through the Safely Remove Hardware Window fails. ***Workaround:*** Do not use the Safely Remove Hardware Window.
Known	IPY00006178 (IPY00005995, IPY00006012, IPY00006039, IPY00006044, IPY00006085)	35039 (35009, 35004, 35003, 35002, 34958)		HMP	If any change is made to the system's IP address after HMP installation (e.g. disconnecting network cables), HMP will fail to download unless a system reboot is performed. ***Workaround:*** Reboot the system after any post-HMP installation IP address change.
Known	IPY00006153	35033		HMP	Occasionally, during repetitive downloads of 240 channel configurations, HMP download will fail. ***Workaround:*** Repeat download.
Known	IPY00006082	34176		HMP	If a kill task with error code 0x3901f occurs, the system requires rebooting before HMP can be restarted successfully. ***Workaround:*** Reboot system
Known (permanent)	--	--		HMP	High I/O activity during heavy HMP activity may cause an increase in error rates, such as degraded digit detection and voice quality.
Known (permanent)	--	--		HMP	HMP may not run properly on a computer if Advanced Configuration and Power Interface (ACPI) is installed. See the *Apic Timer and HMP* information in the *Compatibility Notes* section.
Known (permanent)	--	--		HMP	The HMP system does not operate with Dialogic® board level products, or with System Release software installed on the same machine.
Known (permanent)	--	--		HMP	When using HMP, advanced power management features, such as hibernation, will not be supported.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known	IPY00030768 (IPY00006040, IPY00006047)	36511 (34690, 34255)		Installation	A warning message from the operating system about HMP drivers not being digitally signed may be displayed during installation. ***Workaround:*** Ignore the warning and continue installation. For information on disabling the warning message, please refer to Section 4.5, "Disabling the Windows Driver Signing Check" in the HMP Software Installation Guide.
Known	IPY00006040 (IPY00006047)	34690 (34255)		Installation	A warning message from the operating system about HMP drivers not being digitally signed may be displayed during installation. ***Workaround:*** Ignore the warning and continue installation. For information on disabling the warning message, please refer to Section 4.5, "Disabling the Windows Driver Signing Check" in the HMP Software Installation Guide.
Known	IPY00006308	36489		IP	The Event Viewer shows an information message during startup stating that the default IP address has changed, even though it has not changed. ***Workaround:*** None. Ignore this message.
Known	IPY00006275	36265		Licensing	Macrovision LMTOOLS utility may fail on machines running Windows* XP Service Pack 2 if the System Settings tab is selected. ***Workaround:*** Turn DEP off. Refer to the “Turning DEP Off for the LMTOOLS Utility (Windows XP SP2)” procedure in the LMTOOLS Reference chapter of the Administration Guide.
Known	IPY00006259	36149		Licensing	The Impath utility does not work correctly, preventing CLI license activation from working as documented. ***Workaround:*** Use the utilities described in the Command Line Interface (CLI) Reference chapter of the Administration Guide instead.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known	IPY00006174	36261		Licensing	When using the Macrovision LMPATH utility CLI, download may fail with the following error messages: \Device\SSP, LBRAC, 800C8, qkernerr.h, 800C9, qkernerr.h ***Workaround:*** Use the utilities described in the Command Line Interface (CLI) Reference chapter of the Administration Guide instead.
Known	IPY00006305	36495		Station Interface	The Station Interface API function **si_GetDisplayCursorPos()** does not fail if a station does not have a carrier. ***Workaround:*** None
Known	IPY00006093	34242		Voice	Under heavy load, the TDX_UNDERRUN event may not occur when stream data is played out. ***Workaround:*** None is required. The application should recover.
Known (permanent)	--	--		Voice	DTMF digits not processed by the user application with **dx_getdig()** or other means remain in the device after it is closed with **dx_close()**, and remain with the device even after the application terminates. To ensure that the buffer is empty, clear the digit buffer using **dx_clrdigbuf()**.
Known (permanent)	--	--		Voice	For both single and dual tone definition, the frequency deviation that is defined in the tone template for each frequency must be not less than ± 30 Hz.
Known (permanent)	--	--		Voice	Global Tone Detection (GTD) does not support detecting user defined tones as digits via the digit queue; the tones are only detected as tone events via the event queue.
Known (permanent)	--	--		Voice	The **dx_dial()** function only detects CED tones for CR_FAXTONE event. CNG tones cannot be detected.
Known (permanent)	--	--		Voice	The **dx_getctinfo()** and **fx_getctinfo()** functions return incorrect values for Device Family and Bus Mode.
Known (permanent)	--	--		Voice	The **dx_playtoneEx()** function will not terminate after the time specified by the DX_MAXTIME termination parameter has elapsed.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

Issues Sorted by Type, Dialogic® HMP Software Release 2.0WIN

Issue Type	Defect No.	PTR No.†	SU No.	Dialogic® Product(s) or Component(s)	Description
Known (permanent)	--	--		Voice	The **dx_reciottdata()** and **dx_playiottdata()** functions do not support recording or playing in WAVE format to/from memory; only VOX format.
Known (permanent)	--	--		Voice	The number of tone templates which can be added to a voice device and enabled for detection is limited. The maximum number of events for each instance is 10.

†Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding Change Control System (CCS) defect number are shown. For issues reported after March 27, 2006, this column contains "--" and only the CCS number is used to track the issue.

2.2 Compatibility Notes

Adjusting the TimedWait State

Running ONLY call control on 10 or more timeslots may cause the following error:

"IPEC_Q931Cause18NoUserResponding"

Each IP call uses a Windows socket which binds the call to a unique TCP address/port. The Global Call stack uses these ports starting at port address 20000. When an IP call is completed, the socket associated with that call closes and then enters into a TimedWait state, during which the socket's associated address/port is not available for use until the time expires. The default time for this TimedWait state is 240 seconds.

If an application is stopped and then immediately restarted before the TimedWait state expires, as may be the case during application development and debugging, calls may fail. Reducing the duration of the TimedWait state can alleviate this problem.

Another problem that may result from the TimedWait state duration is when a server experiences a high call rate. Even though the maximum number of TCP connections that can be opened simultaneously is large, in a high call rate scenario the potential exists for hundreds of TCP sockets to be in the TimedWait state causing the system to reach the maximum number of TCP connections. Again, reducing the duration of the TimedWait state can alleviate this problem.

Changing the TimedWait state to a value less than the 240 second default requires a change to the registry:

System Key:	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters
Parameter Name:	TcpTimedWaitDelay

Value Type:	REG_DWORD (DWORD Value)
Valid Range:	30 - 300 seconds

Also, see the following Microsoft information at these links:

- Windows XP *http://support.microsoft.com/default.aspx?scid=kb;en-us;314053*
- Windows 2003 Server *http://support.microsoft.com/default.aspx?scid=kb;en-us;120642*

APIC Timer and HMP

HMP uses the Advanced Programmable Interrupt Controller (APIC) timer for its high resolution timer. The APIC timer operation may conflict with certain modes of the Advanced Configuration and Power Interface (ACPI). During the installation process, the install program checks to determine if the ACPI is enabled. If ACPI is enabled and does not meet the minimum version requirement, a message is displayed stating that the local APIC timer is incompatible with this release and that the setup program will disable the HMP APIC driver.

After the HMP software has been installed and it has been determined that the ACPI is enabled, you must determine if the ACPI is incompatible with HMP by performing the following:

1. Start the HMP media subsystem through the Configuration Manager (DCM). (Refer to the Dialogic Host Media Processing Software Release 2.0WIN Software Installation Guide for information about using DCM.)
2. Start the IP Media Server (Global Call) Demo. (Refer to the IP Media Server (Global Call) Demo Guide for information about starting the demo.)
3. Launch Microsoft* NetMeeting* from another computer, with a sound card, connected on the same LAN.
4. Make sure that the Outgoing Calls security option in NetMeeting (Tools>Options>Security) is unselected.
5. Make sure the default coder in NetMeeting is set to "CCITT u-Law, 8.000 kHz, 8 Bit, Mono". This is set via the Advanced option in the Audio property sheet accessed via Tools>Options.
6. From NetMeeting, connect to the machine running HMP (by typing the IP address or computer name and clicking the telephone icon) and identify if the audio is acceptable.
7. If the audio is acceptable, no further action is necessary. If there is no audio or poor audio, then you must completely disable ACPI in order to run HMP on this system. Disabling ACPI requires that you re-install Microsoft Windows. Microsoft Knowledge Base Article # Q237556 provides instructions for installing Windows without automatically enabling some form of ACPI.

Echo Cancellation With Continuous Speech Processing

For HMP Continuous Speech Processing (CSP), the Echo Cancellation parameter default value is set to OFF (echo cancellation disabled). For IP connections, this parameter must not be turned on by the application. For PSTN connections using the DNI300TEPHMP and DNI1200TEPHMP boards, you may enable echo cancellation.

For information about enabling echo cancellation on HMP, see Section 4.2, “Configuration” in the *Dialogic® Host Media Processing Software 2.0WIN Release Guide*.

For additional information about CSP, refer to the *Continuous Speech Processing API for Linux and Windows Library Reference* and the *Continuous Speech Processing API for Linux and Windows Programming Guide*.

Echo Cancellation With Conferencing

For HMP conferencing, the Echo Cancellation parameter default value is set to OFF (echo cancellation disabled). For IP connections, this parameter must not be turned on by the application. For PSTN connections using the DNI300TEPHMP and DNI1200TEPHMP boards, you may enable echo cancellation.

For information about enabling echo cancellation on HMP, see Section 4.2, “Configuration” in the *Dialogic® Host Media Processing Software 2.0WIN Release Guide*.

For additional information about Conferencing, refer to the *Audio Conferencing API for Linux and Windows Operating Systems Library Reference* and *Audio Conferencing API for Windows Operating Systems Programming Guide*.

Configuring UDP/RTP Port Range

Note: In addition to the following procedure, you may also use the structure IPM_PARM_INFO, associated with the **ipm_GetParm()** and **ipm_SetParm()** API functions, to configure the UDP/RTP port range. For more information, refer to Chapter 4, “Data Structures” in the *IP Media Library API for Host Media Processing Library Reference*.

The HMP system currently defaults to UDP/RTP ports 6000 through 6100 and UDP ports 49152 through 49xxx for RTP streaming, where 49xxx = 49152 + twice the maximum number of channels purchased under the licensing agreement. For example, if 120 channels were purchased, 49xxx would be 49392.

If the UDP/RTP port range used by the HMP system conflicts with other RTP service, the following procedure describes how to configure HMP to use a different UDP/RTP port range:

1. Stop the system service using DCM.
2. Locate the .config file in the *C:\Program Files\Dialogic\HMP\data* directory that matches the FCD/PCD files associated with your licensed configuration.
3. Using a text editor such as NotePad or WordPad, open the .config file.
4. Locate the [IPVSC] section in the .config file.

5. In the [IPVSC] section, locate the line

    ```
     setparm 0x4005, 49512   !set the rtpPortBase on IPVSC
    ```

    The number 49512 is the default value for this parameter. You may change the beginning of the UDP/RTP port range by first editing this value and saving the .config file.
6. After you have saved the .config file with the new UDP/RTP port value, open the Command Prompt window.
7. From the Command Prompt, change the directory to *C:\Program Files\Dialogic\HMP\data*.
8. Execute fcdgen as follows:

    ```
    ..\bin\fcdgen -f <input filename>.config -o <output filename>.fcd
    ```

    The resulting FCD file is created in the *C:\Program Files\Dialogic\HMP\data* directory. If the -o option is omitted from the command, the default output FCD file will have the same filename as the user-modified input .config file, but with an .fcd extension.
9. Restart the system service using DCM to download the new configuration to HMP.

Configuring T.38 Service Port Range

The HMP system currently defaults to port 6000 as the starting UDP/RTP port for the T.38 service port. If the T.38 service port range used by the HMP system conflicts with other RTP service, the following procedure describes how to configure HMP to use a different T.38 service port range:

1. Stop the system service using DCM.
2. Locate the .config file in the *C:\Program Files\Dialogic\HMP\data* directory that matches the FCD/PCD files associated with your licensed configuration.
3. Using a text editor such as NotePad or WordPad, open the .config file.
4. Locate the [0xe] section in the .config file.
5. In the [0xe] section, locate the line:

    ```
    setparm 0x4c21, 6000 ! QFC3_PrmIPRxPortBase (QFC3_PrmLocalUDPPortBase)
    ```

    The number 6000 is the default value for this parameter. You may change the beginning of the T.38 service port range by first editing this value and saving the .config file.
6. After you have saved the .config file with the new T.38 service port value, open the Command Prompt window.
7. From the Command Prompt, change the directory to *C:\Program Files\Dialogic\HMP\data*.
8. Execute *fcdgen* as follows:

    ```
    ..\bin\fcdgen -f <input filename>.config -o <output filename>.fcd
    ```

The resulting FCD file is created in the *C:\Program Files\Dialogic\HMP\data* directory. If the -o option is omitted from the command, the default output FCD file will have the same filename as the user-modified input .config file, but with an *.fcd* extension.

9. Restart the system service using DCM to download the new configuration to HMP.

H.323 Coder Negotiation with Third Party Stacks

Use caution when restricting coder frame sizes or frames per packet while communicating with third-party H.323 stacks. The IPT H.323 protocol stack uses both coder type and frames per packet as part of the algorithm to determine a successful Tx/Rx media match. Restricting coder frame sizes can cause Fast Start calls to fallback to Slow Start or coder negotiation to fail. See the following example.

Example:

A Global Call application configures a channel for G.711 A Law with a frame size of 10 milliseconds. A third-party H.323 stack, which is configured for G.711 A Law with a frame size of 30 milliseconds, initiates a call to the application. The IPT H.323 protocol stack will use 10 milliseconds as an upper limit for both the Tx and Rx media directions. Even though both sides support the same coder, the frame size discrepancy can cause the coder negotiation to fail, resulting in a GCEV_DISCONNECTED event.

SIP Call Control

If the remote site does not respond to an outgoing INVITE sent from HMP, the **gc_MakeCall()** function will time out after 32 seconds and generate a GCEV_DISCONNECTED event. In this scenario, if the application attempts to drop the call before the 32 second timeout is reached, a CANCEL will be sent by HMP to the remote site. If there is no response by the remote site to the CANCEL, there will be an additional 32 second timeout, at the end of which, a GCEV_DISCONNECTED event will be reported.

Documentation Updates 3

This chapter contains information on updates and corrections to the documents included in Dialogic® Host Media Processing Software 2.0WIN. Documentation updates are divided into the following categories.

The documentation updates are divided into the following sections, which correspond to the top level categories used in the online document navigation page:

3.1 Release Documentation Updates

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

- Dialogic® Host Media Processing Software Release for 2.0WIN Release Guide

3.1.1 Dialogic® Host Media Processing Software Release for 2.0WIN Release Guide

Update to **Chapter 3, “Interface Board Support”**
A maximum of 360 full duplex channels are supported by the system, rather than 240 channels.

Update to **Section 4.2, "Configuration"** (IPY00033972)
The following information about configuring parameters for IP media devices in the [IPVSC] section should be added:
Fixed Latency Mode
Number: 0x1b8b
Description: This parameter defines the mode of operation for the jitter buffer. By default, the jitter buffer uses the adaptive mode, which means latency can grow until the available number of frames is depleted. Using the fixed mode, the number of frames buffered can grow up to 1.5 times the starting value (defined by the Initial Latency parameter).
Values:
0 [default] -- adaptive mode enabled
1 -- fixed mode enabled
Guidelines: Use the adaptive mode to minimize audio loss due to abnormal packet reception conditions. Use the fixed mode to minimize latency growth by sacrificing

audio quality. This parameter is not included in the configuration file. To modify the parameter value, you must add this parameter manually in the configuration file in the [IPVSC] section.
Initial Latency
Number: 0x1b07
Description: This parameter specifies the amount of packet loss recovery (PLR) latency (delay) that can be introduced by defining the initial number of frames that can be buffered. This parameter defines the starting value for initial latency.
When the fixed mode is enabled, the number of frames buffered can grow up to 1.5 times the starting value. For example, if the parameter value is 6, the maximum number of frames that can be buffered is 9.
The packet loss recovery module attempts to restore packets arriving at the receive end as close as possible to their original time-stamped positions. Arriving packets are decomposed into individual frames, each with a unique time stamp.
Each new frame is then stored in a jitter buffer before sending it to the decoder. This is done to allow packets arriving out of order to be inserted in the queue in the correct order.
Values:
1 to 200 frames. Default value is 6.
Guidelines: This parameter is not included in the configuration file. To modify the parameter value, you must add this parameter manually in the configuration file in the [IPVSC] section.

Update to **Section 2.2, "Basic Hardware Requirements"** (PTR# 36556):

The following information is added:

- Systems with more than 4 Gigabytes of RAM are not supported.
- Physical Address Extensions (PAE) is not supported. Because systems with less than 4 Gigabytes of memory may use 32-bit addresses, you may not need to disable PAE. If you decide, however, to disable PAE, perform the following:

1. Determine if PAE is enabled by selecting Start -> Control Panel -> System. In the General tab, note if "Physical Address Extensions" is listed under Computer:

2. If "Physical Address Extensions" is listed, PAE is enabled. To disable PAE, proceed to step 3.

3. Edit the c:\boot.ini file by removing the options that begin with "/NOEXECUTE".

4. Add the "/EXECUTE" and "/NOPAE" options to the boot.ini file.

Notes:1. The above procedure also disables Data Execution Prevention (DEP). DEP needs to be disabled because when DEP is enabled, Windows* also enables PAE on machines that are capable of supporting PAE.

2. When PAE is enabled, Windows* may use 64-bit pointers instead of 32-bit pointers, depending on how much memory the machine has.

Updates to **Section 3.1, "New Features in This Release"**
In the **Host Streaming Interface** section, the last paragraph should read:

“For additional information, see the *Dialogic® Digital Network Interface Boards Configuration Guide*.”

The following information applies to the **“Dialogic® Digital Network Interface Boards”** section:

The third bullet under “Key features of the digital network interface boards include:” should read:

- Supports the Dialogic API libraries including audio conferencing, continuous speech processing, device management, digital network interface, Global Call, IPML, standard runtime library, and voice.

The following trunk configuration information is added (PTR# 36733):

Trunk Configuration
The trunks on an Dialogic® Digital Network Interface board can be configured for both the media load and, on a trunk-by-trunk basis, either a T1 or E1 protocol. Trunk configuration can be accomplished either through the Configuration Manager (DCM) utility using the Trunk Configuration Property sheet or programmatically using the NCM API function **NCM_ApplyTrunkConfiguration( )**.

For information about configuring the trunks using DCM, see the DCM Online Help. For information about configuring the trunks programmatically, refer to the Native Configuration Manager API for Windows Operating Systems Library Reference and the Native Configuration Manager API for Windows Operating Systems Programming Guide.

The following API information is added:

For Dialogic® Network Interface Boards, the Global Call API is used to provide call control functionality on PSTN interfaces. For E1, T1 and ISDN technologies, the libdm3cc.dll library provides this functionality and is dynamically loaded, by specifying GC_DM3CC_LIB when calling the **gc_Start( )** function.

In the **“Dialogic® Digital Network Interface Boards section,”** the last paragraph should read:

“For more information about media loads and how to configure the digital network interface boards, see the *Dialogic® Digital Network Interface Boards Configuration Guide* and the Configuration Manager (DCM) online Help.”

The following API information applies to the **“Dialogic® Digital Station Interface Boards”** section:

For Dialogic® Digital Station Interface Boards, the Station-Side Interface API should be used to provide call control processing and station-side application programming interfaces for proprietary digital station sets. The libdm3cc.dll library provides this functionality.

Update to **Section 4.2, “Configuration”**

The third paragraph should read:

> “For detailed configuration information, see the DCM online context-sensitive Help and the *Dialogic® Digital Network Interface Boards Configuration Guide*.”

Update to **Section 9.1, “Documentation Support for HMP 2.0WIN Features”**:

> In Table 6, the part number for Digital Network Interface Boards Configuration Guide is now 05-2474-002.

3.2 Installation and Configuration Documentation Updates

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

- Dialogic® Host Media Processing Software Release 2.0WIN Software Installation Guide
- Dialogic® Digital Network Interface Boards Configuration Guide
- Dialogic® Global Call CDP Configuration Guide

3.2.1 Dialogic® Host Media Processing Software Release 2.0WIN Software Installation Guide

Update to section 2.1, "Prerequisites for Software Installation," page 13

> The following requirement should be added to the list, after Windows operating system:
>
> - The Dynamic Host Configuration Protocol (DHCP) Windows Client Service which is part of the Windows operating system must be enabled in order to obtain the IP address before the HMP system starts. It is also recommended that the DHCP Client startup type be set to "Automatic." If it is set to manual mode, the HMP system will not start until DHCP service is started manually.

Update to section 2.3, "Performing a Full Install of the Software," page 23

> In the bullet about checking the IP address, the following information should be added:
>
> - The IP address used by HMP is obtained by the DHCP Client Windows Service before the HMP system starts. Be sure DHCP Windows Service is enabled on your system.

Update to section 2.4, "Performing an Update Install," page 25

> In the bullet about checking the IP address, the following information should be added:
>
> - The IP address used by HMP is obtained by the DHCP Client Windows Service before the HMP system starts. Be sure DHCP Windows Service is enabled on your system.

Update to section 2.5, "Performing a Silent Install," page 28-29

In the bullet about checking the IP address, the following information should be added:

- The IP address used by HMP is obtained by the DHCP Client Windows Service before the HMP system starts. Be sure DHCP Windows Service is enabled on your system.

3.2.2 Dialogic® Digital Network Interface Boards Configuration Guide

Update to **Section 6.8, "Trunk Configuration Property Sheet"**

Group 2 protocol, DPNSS (E1) is now supported.

Update to Chapter 7, "CONFIG File Parameter Reference" (PTR: 36210)

The Dialogic® Digital Network Interface Boards Configuration Guide does not include information about the PVD and PAMD qualification templates that are defined in the CONFIG file. The relevant parameters are in the [SigDet] section of the CONFIG file.

In addition, the default PVD and PAMD qualification template definitions can be modified in accordance with the instructions in Technical Note 030 available on the Customer Support web site at:
http://www.dialogic.com/support/

Note: Technical Note 030 is not specifically written for HMP, but the same principle applies.

The recommended default PVD (ID=128193) and PAMD (ID=106561) qualification template definitions are as follows:

```
!User defined Pvd template.
PvdDesc signalId 128193
PvdDesc signalLabel 0000
PvdDesc minSnr 5
PvdDesc maxSnr 600
PvdDesc maxPk 2
PvdDesc maxRing 5
PvdDesc ringThresh 10000
PvdDesc PvdWin 8
PvdDesc PvdVthresh 10000
PvdDesc PvdRbLow 380
PvdDesc PvdRbHigh 510
CreatePvd

!User defined PAMD template.
PamdDesc signalId 106561
PamdDesc signalLabel 0000
PamdDesc minRing 190
PamdDesc mask 1
PamdDesc maxAnsiz1 159
PamdDesc maxAnsiz2 159
PamdDesc maxAnsiz3 159
PamdDesc loHiss 22
PamdDesc hiHiss 16
PamdDesc bhParm 5
PamdDesc cvThresh1 80
PamdDesc cvThresh2 165
PamdDesc maxCvThresh 390
PamdDesc nMaxBroad 2
PamdDesc nMaxErg 65
PamdDesc maxSilence 30
PamdDesc voiceThresh 25
PamdDesc silenceThresh 10000
PamdDesc rjFbandLow 0
PamdDesc rjFbandHigh 0
CreatePamd
```

The default PVD qualification template ID is 128193 (0x1f4c1), but other valid PVD qualification template IDs that can be defined in the CONFIG file are:

- 128194 (0x1f4c2)
- 128195 (0x1f4c3)
- 128196 (0x1f4c4)
- 128197 (0x1f4c5)

The default PAMD qualification template ID is 106561 (0x1a041), but other valid PAMD qualification template IDs that can be defined in the CONFIG file are:

- "106564 (0x1a044)
- "106565 (0x1a045)
- "106566 (0x1a046)
- "106567 (0x1a047)

3.2.3 Dialogic® Global Call CDP Configuration Guide

Update to **Section 2.4.2, "Downloading the Protocol and CDP File on a Windows System"** (IPY00031524 = PTR# 36868)

When setting up the pdk.cfg file for DNI601TEPHMP dual span interface boards and DNI1200TEPHMP quad span interface boards, an mlmfile option must be specified. This section currently says to use **mlmfile hmp_media_pdk.mlm.sym** for DNI601TEPHMP boards and **mlmfile hmp_pdk.mlm.sym** for DNI1200TEPHMP boards. This is correct, but only if the board is using CAS or R2MF on all trunks.

If the board is mixing CAS/R2MF and ISDN protocols, use the following mlmfile options:

- **mlmfile hmp_media_mixed.mlm.sym** for DNI601TEPHMP boards
- **mlmfile hmp_mixed.mlm.sym** for DNI1200TEPHMP boards

When a board is mixing CAS/R2MF and ISDN protocols, the *pdk.cfg* file should indicate the trunks that are configured for CAS/R2MF. For example, if a

DNI1200TEPHMP board is configured to have two 4ESS trunks and two CAS trunks, then the *pdk.cfg* file should contain:

```
b 0 line 3 v pdk_us_mf_io.cdp p gnetworkonly_hmpqsb_2_4ess_2_cas.pcd
f gnetworkonly_hmpqsb_2_4ess_2_cas.fcd m hmp_mixed.mlm.sym r 1
b 0 line 4 v pdk_us_mf_io.cdp p gnetworkonly_hmpqsb_2_4ess_2_cas.pcd
f gnetworkonly_hmpqsb_2_4ess_2_cas.fcd m hmp_mixed.mlm.sym r 1
```

3.2.4 Dialogic® DCM Online Help

Update to the DCM Online Help Trunk 1 through Trunk 4 Help topics:

The Note in the Trunk 1 through Trunk 4 parameter topics should be:

Note: Depending on the board, you may assign the same protocol or different protocols to each trunk on the board, but all protocols must belong to the same group:

- For the DNI300TEPHMP board, you may assign an ISDN protocol or the CAS protocol to the trunk.
- For the DNI601TEPHMP board, you may assign an ISDN, CAS, or R2MF protocol to either trunk.
- For the DNI1200TEPHMP board, you may assign an ISDN protocol or the CAS protocol to any trunk.

3.3 OA&M Documentation Updates

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

- Dialogic® Host Media Processing Administration Guide
- Dialogic® SNMP Agent Software Administration Guide
- Dialogic® Host Media Processing Diagnostics Guide
- Event Service Programming Guide
- Event Service Library Reference
- Dialogic® Native Configuration Manager API for Windows Operating Systems Programming Guide
- Dialogic® Native Configuration Manager API for Windows Operating Systems Library Reference

3.3.1 Dialogic® Host Media Processing Administration Guide

Update to **Section 2.1, “Preparations”**

In Table 3, the word “Note:” should be removed from the Description of the RTP feature.

Update to **Section 2.4, Obtaining a Trial, Purchased, or Emergency Replacement License"**

The following information applies to the **"Obtaining a License File that is Locked to a Machine"** section:

A second Caution note should be added:

Caution: Certain changes to the machine and its environment could cause the lock code to change. If this occurs, a new license will be required. This is most likely to occur on machines with multiple network cards. Conditions which might cause the lock code to change are: updates to network card drivers, disabling or uninstalling certain network cards, OS service pack updates, OS upgrade or reinstalls, and computer name changes.

The following note should be added:

Note: If your system has multiple NIC cards, you can avoid having the lock code become invalid by using the procedure provided at the end of "System Fails to Start on Boardless System, License Does Not Work," which is found in Chapter 12, "Troubleshooting."

The following note should be added:

Note: A license that is locked to a machine will not work if an Dialogic DNI or DSI board is physically in the system (even if the DNI or DSI board software is not installed). You must remove the board(s) before you can use a license that is locked to a machine.

In Step 1, the link should be changed:

Change the link for obtaining *LicenseManagerNSI* utility to the following: *http://www.dialogic.com/support*

Update to **Section 3.1, "Requirements"**

The following Note should be added:

Note: A license that is locked to a machine will not work if an Dialogic DNI or DSI board is physically in the system (even if the DNI or DSI board software is not installed). You must remove the board(s) before you can use a license that is locked to a machine.

Update to **Section 3.4, "Activating a License with the Intel® VTune™ Performance Analyzer Installed"**

The following changes apply:

There is a typo in Step 5.

The command `lmutil ilmpath -status` should be `lmutil lmpath -status`.

Step 6 has been clarified and expanded to the following:

Delete the existing HMP PCD, FCD, and Config files. They are in *c:\Program Files\Dialogic\hmp\data*. The file extensions are *.pcd*, *.fcd*, and *.config*.

Only the HMP PCD, FCD, and Config files should be deleted. The hardware/board PCD, FCD, and Config files should not be deleted. The hardware/board PCD, FCD, and Config files begin with one of the following:

- *hmp_ssb_default*
- *hmp_dsb_default*
- *hmp_qsb_default*

The names of the HMP PCD, FCD, and Config files that should be deleted will start with one of the following:

- *1r1v0e0c0s0f_ver*
- another string with *<number>r<number>v<number>e<number>c<number>s<number>f_ver*
- the name of the file into which you pasted the contents of the HMP license file

This will be the file listed by the lmutil lmpath –status command.

Update to **Section 4.1, "Requirements"**

The following Note should be added:

Note: A license that is locked to a machine will not work if an Dialogic DNI or DSI board is physically in the system (even if the DNI or DSI board software is not installed). You must remove the board(s) before you can use a license that is locked to a machine.

Update to Section 7.4, "Stopping HMP". Add the following information after item 2:

When HMP is stopped via DCM, the Dialogic System Service (DSS) and all other Dialogic services will be stopped. When HMP is started again via DCM, the DSS and all other Dialogic services will be started automatically.

Note: The Boardserver service (BoardServer.exe service for the SNMP Agent Software functionality) is not a part of the DCM start-up sequence. If the Boardserver service was running prior to stopping HMP, Boardserver must be restarted manually. If the Boardserver service is set to start automatically when the system starts, it will start automatically for the new session.

Note: The **NCM_StopSystem( )** function cannot be used to stop Dialogic services. This function is intended to stop both the HMP virtual device or a physical device.

Updates to **Chapter 11, "Troubleshooting"**

In the **"License Fails with Boards"** section, Step 2 should read:

Run

```
lmutils lmpath -status.
```

Add the topic, **Errors Using HMP with RealSpeak 4.0** (IPY00035708).

Using HMP 2.0 and RealSpeak 4.0 software in the same chassis may result in an error because both products use the Flexnet Licensing System from Macrovision Corp. Because the port IDs are not specified for each system, the Flexnet Licensing System automatically uses the first available TCP/IP port in the range of 27000 – 27009. An error results because HMP 2.0 and RealSpeak 4.0 systems are assigned to use the same first 27000 port.

To avoid this problem, modify the HMP licensing file to assign an alternative TCP/IP port explicitly for HMP licensing. The following example shows the modified licensing file. Notice that the first line specifies a 270007 port number which will be used for the HMP licensing.

```
SERVER this_host ANY 270007
VENDOR INTEL
FEATURE Voice INTEL 3.0 permanent 60 \
VENDOR_STRING=LIC_TYPE=Purchased,APP=HMP,HOSTID=3000E0C2DE3F6 \
HOSTID=000e0c2de3f6 vendor_info=Vendor_Info TS_OK \
SIGN=E23EFAF66198
FEATURE Enhanced_RTP INTEL 3.0 permanent 60 \
VENDOR_STRING=LIC_TYPE=Purchased,APP=HMP,HOSTID=3000E0C2DE3F6 \
HOSTID=000e0c2de3f6 vendor_info=Vendor_Info TS_OK \
SIGN=474711B09D54…
```

The following subsection should be added:

System Fails to Start on Boardless System, License Does Not Work

In this situation, you will get an error message similar to the following:

```
General fault: MC_ERROR_FEATURE_QUERYCHECKOUT:  Failed to get the list
of features from the License Manager
 Board Number:       0
 Processor:              0
 Instance:  0
Additional Data: FlexLM Function 'lc_checkout()' returned error: (-9)
Invalid host.
 The hostid of this system does not match the hostid specified in the
license file.
Feature:       RTP_G_711
Hostid:        COMPOSITE=1DCD62854D81
```

This error may occur if the system has more than one NIC card and the role of primary (default) NIC card has been reassigned to a different NIC card. If a machine has multiple NIC cards, the license is always tied to the “default NIC.” If the default NIC changes for some reason between the time you run the NSI utility to get the lock code for the license and the time you activate the license, the license becomes invalid. One reason the role of default NIC might be reassigned is if the default NIC becomes disabled.

This problem can also be caused if the network name of the HMP computer is changed. Since the host ID is a combination of the NIC card ID and computer network name, you should determine whether the network name of the HMP computer has been changed.

To determine whether this is the cause of the problem, run the NSI utility and verify that the lock code returned is the same as in the license file (use of the NSI utility is described in Obtaining a License File that is Locked to a Machine).

To resolve this problem, you need to disable all NIC cards except one and generate a license specifically for the enabled NIC. After the license is activated based on this enabled NIC card, you can enable all other NIC cards. Here is the procedure:

1. Disable all NIC cards except one (the primary NIC).
   You can do this via the Network Connections panel.
2. Follow the procedure in 'Obtaining a License File that is Locked to a Machine.'
3. After you activate the license, enable the NIC cards you disabled.

3.3.2 Dialogic® SNMP Agent Software Administration Guide

There are currently no updates to this document.

3.3.3 Dialogic® Host Media Processing Diagnostics Guide

Update to **Section 5.2, "Collecting System Data to Diagnose an Application Failure or Crash"** and **Section 5.3, "Creating a System Configuration Archive"**

Change the Windows procedure for starting its_sysinfo to the following:

- Windows systems:

1. Click on *its_sysinfo.exe* in %INTEL_DIALOGIC_DIR%\bin.
2. Click the **Generate** button. A dialog box appears on which you must name the archive file into which you want the information to be collected. The default filename is *its_sysinfo.zip*.
3. Click the **Save** button and its_sysinfo will start collecting information.
4. A pop-up window displaying “Data collection completed. Zip archive was created as <zip file name>” will appear to indicate completion of *its_sysinfo.exe* execution. Click the **OK** button. The archive file is in the location specified in the tool.

Note: In addition to the procedure change, the name of the executable has changed from *its_sysinfo_gui.exe* to *its_sysinfo.exe*.

3.3.4 Event Service Programming Guide

There are currently no updates to this document.

3.3.5 Event Service Library Reference

There are currently no updates to this document.

3.3.6 Dialogic® Native Configuration Manager API for Windows Operating Systems Programming Guide

There are currently no updates to this document.

3.3.7 Dialogic® Native Configuration Manager API for Windows Operating Systems Library Reference

Added two parameters to the **NCM_ApplyTrunkConfiguration( )** function (IPY00032809):

pMediaLoad	pre-defined sets of supported features. A media load consists of a configuration file set and associated firmware. Universal media loads support voice, fax, and conferencing resources simultaneously.
pErrorMsg	API provides as needed.

Added the following example code for a DMV/B series quad span board (i.e., DMV1200BTEP):

```
#include <stdio.h>
#include "ncmapi.h"
void main()
{
NCMRetCode ncmRetCode;
char buffer[300] = {0};

         NCMFamily family;
         family.name = "DM3";
         family.next = NULL;

         NCMDevice UniqueName;
         UniqueName.name = "DMV1200BTEP #1 in slot 2/10";
         UniqueName.next = NULL;

         NCMTrunkConfig ncmTruckConfig[4] = {0};

         NCMFeatureType ncmFeatureType = {0};


         ncmTruckConfig[0].TrunkName  = "Trunk1";
         ncmTruckConfig[0].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[0].next      = &(ncmTruckConfig[1]);

         ncmTruckConfig[1].TrunkName  = "Trunk2";
         ncmTruckConfig[1].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[1].next      = &(ncmTruckConfig[2]);

         ncmTruckConfig[2].TrunkName  = "Trunk3";
         ncmTruckConfig[2].TrunkValue = "5ESS(T1, Group 1)";
         ncmTruckConfig[2].next      = &(ncmTruckConfig[3]);

ncmTruckConfig[3].TrunkName  = "Trunk4";
         ncmTruckConfig[3].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[3].next = NULL;

         strncpy(ncmFeatureType.MediaLoad, "ML10", MEDIA_LOAD_LENGTH);

         ncmRetCode = NCM_ApplyTrunkConfiguration(family,UniqueName , ncmTruckConfig,
&ncmFeatureType, reinterpret_cast<unsigned char*>(buffer));
         if (ncmRetCode != NCM_SUCCESS)
         {
           printf("Error calling NCM_ApplyTrunkConfiguration(). It returned: %d \n", ncmRetCode;
           printf( " Error Msg:  %s \n", buffer);
         }
              else
              {
                    printf("SUccessful calling NCM_ApplyTrunkConfiguration\n");
              }
              printf("press any key to exit\n");
              getchar();
}
...
```

Update to **NCM_StartDlgSrv( )** function

The second note in the Description section is invalid and should be removed. The function description should read as follows:

The **NCM_StartDlgSrv( )** function initiates the entire Dialogic system and starts all boards. To start only one board, use **NCM_StartBoard( )** function.

Note: A successful completion code for this function (NCM_SUCCESS) only indicates that a start message was sent to the Dialogic system. Use **NCM_GetDlgSrvState( )** or **NCM_GetDlgSrvStateEx( )** to determine whether or not the system actually started.

Update to **NCM_StartSystem( )** function

The note in the Description section is invalid and should be removed. The function description should read as follows:

The **NCM_StartSystem( )** function starts the Dialogic system service and starts all boards.

If your system is running in manual mode, the **NCM_StartSystem( )** function starts the system service and starts all boards in the system. If your system is running in semi-automatic mode, the Dialogic system service will run uninterrupted and a call to the **NCM_StartSystem( )** function will start all boards. Use the **NCM_GetSystemState( )** function to determine whether or not the system service is running.

Update to **NCM_StopDlgSrv( )** function

The second note in the Description section is invalid and should be removed. The function description should read as follows:

The **NCM_StopDlgSrv( )** function stops the entire Dialogic system and stops all boards.

On HMP, the Boardserver service (*BoardServer.exe* for the SNMP Agent Software functionality) will be stopped as well. If the Boardserver service was running prior to stopping the entire Dialogic system, Boardserver must be restarted manually. If the Boardserver service is set to start automatically when the system starts, it will start automatically for the new session. The Boardserver service is not a part of the **NCM_StartDlgSrv( )** or DCM GUI start-up sequence.

To stop only one board, use the **NCM_StopBoard( )** function.

Note: A successful completion code for this function (NCM_SUCCESS) only indicates that this function attempted to stop the system. Use **NCM_GetDlgSrvState( )** or **NCM_GetDlgSrvStateEx( )** to determine whether or not the system was actually stopped.

Update to **NCM_StopSystem( )** function

The note in the Description section is invalid and should be removed. The function description should read as follows:

The **NCM_StopSystem( )** function stops all boards in the system and, and in some cases stops the Dialogic system service.

If your system is running in semi-automatic mode, the **NCM_StopSystem( )** function will stop all boards in the system, but will not stop the Dialogic system service . If your system is running in automatic or manual mode, the **NCM_StopSystem( )** function will stop all boards and stop the Dialogic system service.

3.4 Programming Libraries Documentation Updates

This section contains updates to the following documents:

- Dialogic® Audio Conferencing API Programming Guide
- Dialogic® Audio Conferencing API Library Reference
- Dialogic® CSP API Programming Guide
- Dialogic® CSP API Library Reference
- Dialogic® Global Call API for HMP Programming Guide
- Dialogic® Global Call API Library Reference
- Dialogic® Global Call IP for HMP Technology Guide
- Dialogic® Global Call ISDN Technology Guide
- Dialogic® Global Call E1/T1 CAS/R2 Technology Guide
- Dialogic® Device Management API for Windows and Linux Operating Systems Library Reference
- Dialogic® Fax Software Reference
- Dialogic® IP Media Library API for HMP Library Reference
- Dialogic® SRL API Programming Guide
- Dialogic® SRL API Library Reference
- Station Side Interface API Library Reference
- Dialogic® Voice API for HMP Programming Guide
- Dialogic® Voice API for HMP Library Reference

3.4.1 Dialogic® Audio Conferencing API Programming Guide

Update to Section 6.2, “Initialization of DM3 Board Parameters” and to “Active Talker” chapter (IPY00006584 = PTR 36199)

Changed the description of the MSG_ACTID parameter to indicate that it enables/disables active talker identification (or notification) and not the active talker feature itself. Replaced the MSG_ACTID parameter with the following:

MSG_ACTID (Active Talker Identification)

Enables or disables Active Talker Identification (or Notification). Possible values are ACTID_ON or ACTID_OFF. ACTID_ON is the default. This parameter does not enable or disable the active talker *feature*, which is always enabled. It only disables the *notification* to the application program. The active talker *feature* sums the 3 most active talkers in a conference, so that the conversation doesn’t get drowned out when too many people talk at once. Active talker *notification* provides data on active talkers through the **dcb_gettalkers( )** and **dcb_GetAtiBitsEx( )** functions, which can be used by an application program to identify active talkers; for example, to provide a visual display highlighting the active talkers in a conference. Active talkers are determined by their loudness; i.e., the strength of their “non-silence” energy.

Note: In some cases, it is desirable to deactivate the active talker feature, such as for a background music application program. Although you cannot directly disable the active talker *feature*, you can set the noise level threshold by which signals are recognized as either speech or noise. For more information, see the background music feature in the *Audio Conferencing API Programming Guide*.

Update to “Background Music” chapter (IPY00010946 = PTR 36323)

Changed description of how to implement background music in an application so that it doesn’t refer to MSG_ACTID. Replaced the instructions on how to implement background music with the following:

Do the following to implement background music in a conference:

- Create a three-party conference, where one party is the music resource.
- When you add music to the conference, make the following parameter settings. Set its party attributes so that it uses transmit-only mode. That is, for the conference party that transmits music, enable the MSPA_MODEXMITONLY attribute in the MS_CDT data structure **chan_attr** field.
- The Conferencing AGC Noise Level Lower Threshold may need to be adjusted. This parameter is set at -40 dBm by default and filters out any signals below this level. If the background music levels are low, the music may not be summed into the conference by the active talker feature, or it may come in and out. In this case, the AGC Noise Level Lower Threshold should be reduced by adding or changing the CSUMS_AGC_low_threshold (0x3B1F) parameter in the configuration file for the board and re-starting the board. For more information on this parameter, see the applicable configuration guide; e.g., *Dialogic® on DM3 Architecture Configuration Guide*, or the configuration guide for Dialogic® Host Media Processing (HMP) software.

3.4.2 Dialogic® Audio Conferencing API Library Reference

Update to **dcb_getbrdparm( )** and **dcb_setbrdparm( )** functions (IPY00006584 = PTR 36199)

Changed the description of the MSG_ACTID parameter to indicate that it enables/disables active talker identification (or notification) and not the active talker feature itself. Replaced the MSG_ACTID parameter with the following:

MSG_ACTID (Active Talker Identification)

Enables or disables Active Talker Identification (or Notification). Possible values are ACTID_ON or ACTID_OFF. ACTID_ON is the default. This parameter does not enable or disable the active talker *feature*, which is always enabled. It only disables the *notification* to the application program. The active talker *feature* sums the 3 most active talkers in a conference, so that the conversation doesn’t get drowned out when too many people talk at once. Active talker *notification* provides data on active talkers through the **dcb_gettalkers( )** and **dcb_GetAtiBitsEx( )** functions, which can be used by an application program to identify active talkers; for example, to provide a visual display highlighting the active talkers in a conference. Active talkers are determined by their loudness; i.e., the strength of their "non-silence" energy.

Note: In some cases, it is desirable to deactivate the active talker feature, such as for a background music application program. Although you cannot directly disable the active talker *feature*, you can set the noise level threshold by which signals are recognized as either speech or noise. For more information, see the background music feature in the *Audio Conferencing API Programming Guide*.

Added to DCB Library Reference **dcb_estconf( )** and to **dcb_addtoconf( ) f**unctions (Defect# IPY00010946 = PTR# 36323)

HMP Only: An EDT_HSIBRIDGEERR indicates that an error occurred in creating a conference or adding a party to a conference because parties are on a different bus fabric than the conference and a Host Streaming Interface (HSI) bridge connection could not be created between HMP and the board. You may be able to recover from this error by waiting for a HSI bridge connection to become available when parties are removed and/or conferences are deleted.

Added to DCB Library Reference chapter on Error Codes:

EDT_HSIBRIDGEERR

HMP Only: Error when creating a Host Streaming Interface bridge connection between HMP and a board.

3.4.3 Dialogic® CSP API Programming Guide

There are currently no updates to this document.

3.4.4 Dialogic® CSP API Library Reference

There are currently no updates to this document.

3.4.5 Dialogic® Global Call API for HMP Programming Guide

Update to section 7.2.4, "Setting Call Analysis Attributes on a Per Call Basis"
Revised the descriptions of the CCPARM_CA_PVD_QTEMP and CCPARM_CA_PAMD_QTEMP parameters (PTR 36210). The Correct information is as follows:

CCPARM_CA_PVD_QTEMP

PVD Qualification Template. Specifies which PVD template to use. Possible values are:

• PAMD_QUAL1TMP - Predefined qualification template. This is the default value.

• -1 - No qualification template

The CCPARM_CA_PVD_QTEMP parameter can also be set to a qualification template ID that is defined in the CONFIG file.

Setting CCPARM_CA_PVD_QTEMP to a value of PAMD_QUAL2TMP is **not** supported.

CCPARM_CA_PAMD_QTEMP

PAMD Qualification Template. Specifies which PAMD template to use. Possible values are:

- PAMD_QUAL1TMP - Predefined qualification template. This is the default value.
- -1 - No qualification template

The CCPARM_CA_PAMD_QTEMP parameter can also be set to a qualification template ID that is defined in the CONFIG file.

Setting CCPARM_CA_PAMD_QTEMP to a value of PAMD_QUAL2TMP is **not** supported.

Note: By default, qualification template parameters are set to the most common values. However, it is possible to tune these parameters in the CONFIG file as described in Technical Note 030 available on the Customer Support web site at: *http://www.dialogic.com/support/*
The technical note is not written specifically for HMP, but the same principle applies.

3.4.6 Dialogic® Global Call API Library Reference

The following information applies to the Global Call API Library Reference:

Code examples

The purpose of the code examples in this document is to illustrate how Global Call functions are used. The examples are not necessarily for HMP-specific applications.

Functions supported in async mode only

Many of the Global Call functions are supported in asynchronous mode only when used with HMP. The async only limitation is for IP, not for calls over the PSTN. Refer to the "IP-Specific Function Information" chapter in the Global Call IP for Host Media Processing Technology Guide for any limitations or differences in function support when used with IP technology.

References to different technologies

Global Call functions can be used with many different technologies. The technologies supported in this release of HMP are IP, E1/T1, and ISDN. Throughout this document, HMP users should ignore any references to Analog, SS7, and technologies for Springware boards.

3.4.7 Dialogic® Global Call IP for HMP Technology Guide

The following information applies to Section 3.4 of the Global Call IP for HMP Technology Guide:

Add a call scenario demonstrating the Fax IP/PSTN gateway

- Supporting the T.38 Fax Gateway

Add a description of the gateway

A new gateway capability provides a mechanism to convert one form of fax transmission to another (i.e., T.38 to V.17, and vice versa).

From the application's perspective, using the gateway capability consists of initiating the gateway processing after each side of the gateway is connected to the HMP R4 fax device and enabling an event handler to collect the completion event for the gateway session.

For example, the T.38 endpoint is connected to the R4 Fax device using the IPML APIs or the GC API's gc_Extension model and the PSTN portion is connected to the fax device using the TDM xx_listen APIs.

Note: For an IP call, either inbound or outbound, the call has to be connected and in T38 Mode. For PSTN, the call has to be connected and fx_listen/gc_listen already completed BEFORE fx_initstat with the DF_T38GW parameter is invoked.

Once this is done, the application invokes the R4 Fax function fx_initstat with the newly defined parameter DF_T38GW.

Once the fax gateway processing completes (either successfully or not), the R4 Fax device will generate a new fax event: TFX_T38GW. It is the responsibility of the application to enable an SRL event handler to capture this event and collect the status of the completed operation.

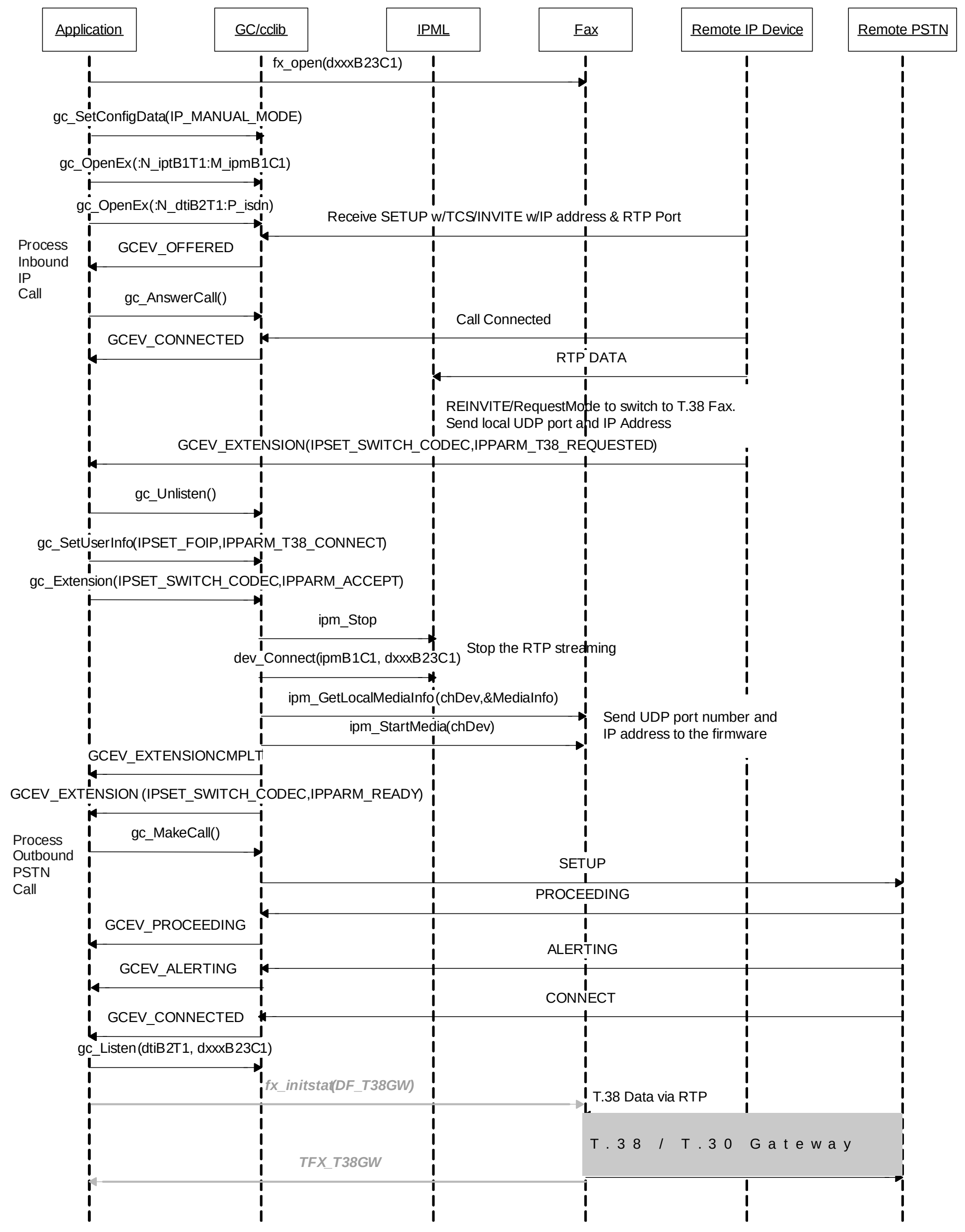
For this release, the application can not actively stop the fax gateway operation. If the application finds it is necessary to terminate the operation, it must do so indirectly by either uncoupling the R4 Fax device from the PSTN or IP endpoints (via the TDM xx_listen APIs) or by using the appropriate call control APIs to terminate either portion of the gateway's calls. The R4 Fax device will then generate the fax event: TFX_T38GW with an error indicating fax under run or timeout.

Call tear down will consist of uncoupling (unlistening) the fax device from the PSTN DTI device and terminating the PSTN call as any normal PSTN call. From the IP perspective, call tear down processing should not differ from current processing.

Add a call flow diagram showing the gateway to the PSTN

(Insert Fax Gateway Call Processing diagram as shown on the next page.)

Fax Gateway Call Processing


Application
GC/cclib
IPML
Fax
Remote IP Device
Remote PSTN
fx_open(dxxxB23C1)
gc_SetConfigData(IP_MANUAL_MODE)
gc_OpenEx(:N_iptB1T1:M_ipmB1C1)
gc_OpenEx(:N_dtiB2T1:P_isdn)
Receive SETUP w/TCS/INVITE w/IP address & RTP Port
Process Inbound IP Call
GCEV_OFFERED
gc_AnswerCall()
Call Connected
GCEV_CONNECTED
RTP DATA
REINVITE/RequestMode to switch to T.38 Fax. Send local UDP port and IP Address
GCEV_EXTENSION(IPSET_SWITCH_CODEC,IPPARM_T38_REQUESTED)
gc_Unlisten()
gc_SetUserInfo(IPSET_FOIP,IPPARM_T38_CONNECT)
gc_Extension(IPSET_SWITCH_CODEC,IPPARM_ACCEPT)
ipm_Stop
Stop the RTP streaming
dev_Connect(ipmB1C1, dxxxB23C1)
ipm_GetLocalMediaInfo(chDev,&MediaInfo)
ipm_StartMedia(chDev)
Send UDP port number and IP address to the firmware
GCEV_EXTENSIONCMPLT
GCEV_EXTENSION (IPSET_SWITCH_CODEC,IPPARM_READY)
Process Outbound PSTN Call
gc_MakeCall()
SETUP
PROCEEDING
GCEV_PROCEEDING
ALERTING
GCEV_ALERTING
CONNECT
GCEV_CONNECTED
gc_Listen(dtiB2T1, dxxxB23C1)
fx_initstat(DF_T38GW)
T.38 Data via RTP
T.38 / T.30 Gateway
TFX_T38GW

The following information applies to Section 4.7 of the Global Call IP for HMP Technology Guide:

Add a new item to the bullet list immediately following the section entitled “Overview of the SIP re-INVITE Method,” as follows:

- Enabling Application Access to re-INVITE Requests

Add a new subsection to section 4.7, “Modifying an Existing SIP Call (re-INVITE)” between the first subsection (“Overview of the SIP re-INVITE Method”) and the second subsection (“Receiving SIP re-INVITE Requests”):

Enabling Application Access to re-INVITE Requests

In order to have access to SIP re-INVITE requests, applications must set a specific parameter value using the Global Call **gc_SetConfigData( )** function. To enable the three **gc_xxxModifyMedia( )** APIs and the GCEV_REQ_MODIFY_CALL event type that is used to notify applications of re-INVITE requests, the application must include the following parameter element in the GC_PARM_BLK that it passes to the **gc_SetConfigData( )** function:

IPSET_CONFIG

IPSPARM_OPERATING_MODE

• value = IP_T38_MANUAL_MODIFY_MODE

Unless this parameter value is set, any attempt to call one of the **gc_xxxModifyMedia( )** functions will fail with an IPERR_BAD_PARM error code.

The following information applies to Section 4.7.1 of the Global Call IP for HMP Technology Guide:

The Global Call implementation of re-INVITE for first party call control only supports the following capabilities:

- specifying, changing, or refreshing header field values or parameters for the existing dialog; for example, refreshing expiring Contact information
- changing the direction of the streaming - changing from half-duplex to full-duplex streaming

The following information applies to Section 4.24.1 of the Global Call IP for HMP Technology Guide:

Add a description of the fax gateway

A new gateway capability provides a mechanism to convert one form of fax transmission to another (i.e., T.38 to V.17, and vice versa).

The conversion is transparent to both the PSTN and IP fax processing that face the gateway. The gateway makes it possible to convert commands and data from one protocol to the other, while maintaining data and protocol integrity. The gateway does not support any image processing of fax data, nor is there any T.30 protocol processing.

In Section 8.3.17, "gc_MakeCall( ) Variances for IP", the last paragraph before Section 8.3.17.1 (page 421) is updated as follows (Defect# IPY00029956 = PTR# 36646):

When using SIP, if the remote side does not send a final response to an outgoing INVITE (sent by the call control library) within 64 seconds, the gc_MakeCall( ) function times out and the library generates a GCEV_DISCONNECTED event to the application. If the application attempts to drop the call before the 64 second timeout is reached, the library's behavior depends on whether a provisional response was received. When no provisional response was received before the application cancels the call, the library cleans up the call immediately. But if a provisional response was received before the application attempts to cancel the call, the library sends a CANCEL to the remote endpoint and generates a GCEV_DROPCALL to the application after it receives the 200OK response to the CANCEL and a 487RequestTerminated response for the original INVITE, or when an additional 32-second timeout expires.

On the reference page for the IP_H221NONSTANDARD data structure (page 507), the descriptions of the three data fields are updated as follows (Defect# IPY00029961 = PTR# 36777):

country_code
The country code. Range: 0 to 255; any value x>255 is treated as x%256.

extension
The extension number. Range: 0 to 255; any value x>255 is treated as x%256.

manufacturer_code
The manufacturer code. Range: 0 to 65535; any value x>65535 is treated as x%65536.

Update to section 9.2.2, "IPSET_CALLINFO"
New parameter IDs have been added to IPSET_CALLINFO so that you can send and receive CPN fields via Global Call over an IP network. See Section 1.4, "Support for 360 Full Duplex Channels", on page 24 for more details.

Update to Section 9.2.4. "IPSET_CONFIG"
Update to Table 46. IPSET_CONFIG Parameter, where a new parameter ID has been added to allow the application to set and get PIIE in an H.323 ALERTING message. See Section 1.2, "Access ALERTING Progress Indicator Information Element", on page 15 for more details.

Parameter ID	Data Type & Size	Description	SIP/ H.323
IPPARM_H323_AUTO_PROGRESS_DISABLE	Type: None Size: 0	Parameter ID in IPSET_CONFIG. This parameter is used to disable H323 automatic PROGRESS message after ALERTING on virtual board.	H.323

3.4.8 Dialogic® Global Call ISDN Technology Guide

Update to **Section 4.12, "Controlling B Channel Status"**
Page 156 includes some extraneous information regarding the use of **gc_WaitCall( )** and **gc_ResetLineDev( )**. The text in this section should read:

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

Update to **Section 8.2.18.2, "Using the GC_MAKECALL_BLK Structure"**
The following Caution should be added on page 205:

When using **gc_MakeCall( )** to make an outbound call, if the origination.address field in the GCLIB_MAKECALL_BLK structure is set to NULL or '\0' (null string), the destination.address_plan and the destination.address_type fields in the

GCLIB_MAKECALL_BLK structure are ignored. This precludes the option of using the **gc_SetCallingNum( )** function to set the origination phone number and specifying a value of NULL or '\0' for the origination_phone_number field in the GCLIB_MAKECALL_BLK structure, when the destination number plan and the destination number type values (as specified in the destination.address_plan and destination.address_type fields in the GCLIB_MAKECALL_BLK structure) must be included in the outgoing message.

Update to **Section 8.2.30, "gc_SetCallingNum( ) Variances for ISDN"**

The following Caution should be added on page 216:

When using **gc_MakeCall( )** to make an outbound call, if the origination.address field in the GCLIB_MAKECALL_BLK structure is set to NULL or '\0' (null string), the destination.address_plan and the destination.address_type fields in the GCLIB_MAKECALL_BLK structure are ignored. This precludes the option of using the **gc_SetCallingNum( )** function to set the origination phone number and specifying a value of NULL or '\0' for the origination_phone_number field in the GCLIB_MAKECALL_BLK structure, when the destination number plan and the destination number type values (as specified in the destination.address_plan and destination.address_type fields in the GCLIB_MAKECALL_BLK structure) must be included in the outgoing message.

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

There are currently no updates to this document.

3.4.10 Dialogic® Device Management API for Windows and Linux Operating Systems Library Reference

The following information applies to the Device Management API for Windows and Linux Operating Systems Library Reference:

The error codes chapter and applicable functions have been updated to ignore the following, yet to be implemented, error codes (Defect# IPY00031581 = PTR# 36839):

- EDEV_INVALIDSTATE
- EDEV_NOTCONNECTED
- EDEV_MM_SUBSYSTEMERR

Multimedia

The multimedia devices and associated multimedia features documented in the Device Management API Library Reference are not supported in this release. Reference to these features occurs throughout the document and should be ignored.

3.4.11 Dialogic® Fax Software Reference

The following information is being added in support of a new fax gateway. The new T.38 gateway provides a necessary link between IP and PSTN transport technologies for HMP 2.0 Windows. This will allow a PC running HMP 2.0 to connect an IP network via T.38 protocol (and a NIC) to the PSTN via the T.30 protocol (and a ThinBlade).

Update to the **fx_initstat( )** function

Update to the function syntax Platform:
add HMP 2.0 as a Platform

Add the following note to the Description
For the DF_T38GW mode, the application *must*:

- connect the IP device to the Fax Channel using the gc_Extension model for connecting the IP data stream to the fax device (or the IPML functions directly)
- connect the PSTN device to the Fax Channel using the appropriate CTBus API calls

before calling **fx_initstat( )**. The application also must enable the event handler to process the fax gateway completion event: TFX_T38GW. This will be the indication to the application that the fax gateway processing has completed.

Add the following State value to the Parameter table:

DF_T38GW	set fax channel to gateway mode

Add the following new Cautions:
Once **fx_initstate( )** is called in gateway mode, the application can not stop the fax gateway operation directly. If either the PSTN or IP calls are disconnected prior to the fax gateway session completing, the fax channel will generate the TFX_T38GW event after fax protocol timers time out.
Upon receipt of the TFX_T38GW event, the application can take what ever action it deems appropriate: complete the tear down of either call (PSTN or IP) change IP media codecs. The fax channel itself, is reset by calling **fx_initstat( )** with the desired parameter prior to the application's next fax operation.

Add the following code example (after the existing example):

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

#include <faxlib.h>

int iptdev; /* PSTN device handles. */
int dtidev; /* PSTN device handles. */
int faxdev; /* Fax channel device handle. */

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

{
 int rc;

  .
  .
  .
/*
 * Open the channel using fx_open( ) and obtain the
 * FAX channel device handle in dev. Use dev for all
 * Fax API calls.
 */

 if( faxdev = fx_open( "dxxxB23C1", NULL )) == -1 )
  {
   /* Error opening device. */
   /* Perform system error handling */
   exit( 1 );
  }

/*
 * Establish an SRL handler for the Gateway completion event
 */

 if( sr_enbhdlr( faxdev, TFX_T38GW, GatewayComplete ) == -1 )
   {
    /* Error enabling handler */
    /* Perform system error handling */
    exit( 1 );
   }


/* Open up Global Call devices
 * - PSTN - the DTI device
 * - IP   - the IPT device
 *
 * Be sure to check for failed de
 */

 if( ( rc = gc_OpenEx( &dtidev, ":N_dtiB2T1:P_isdn", EV_SYNC, NULL ) ) != GC_SUCCESS )
   {
    /* Error opening device. */
    /* Perform system error handling */
    exit( 1 );
   }

 if( ( rc = gc_OpenEx( &iptdev, ":N_iptB1T1:M_ipmB1C1", EV_SYNC, NULL ) ) != GC_SUCCESS )
   {
    /* Error opening device. */
    /* Perform system error handling */
    exit( 1 );
  }


/* !!!!!
 *
 * Please consult the document "Global Call IP for Host Media Processing Technology
 * Guide" on  how to set up and process inbound/outbound T38 calls. Specify the fax
 * device opened using the GC API model or the IPML devconnect APIs to associate the
 * fax device w/the inbound or outbound IP call.
 *
 * Also, utilize the PSTN device to place or answer the PSTN call and use the
 * appropriate xx_Listen commands to connect this same fax device to the PSTN portion
 * of the "gateway'd" call
 *
 * Once both sides of the call have been connected, issue the fx_initstat command
 *
 * !!!!!
 */

/*
 * Set the FAX state to be in V17/T28 Gateway mode
 */
```

```
if( fx_initstat( faxdev, DF_T38GW ) == -1)
  {
   /* Error on device. */
   /* Perform system error handling */
   exit( 1 );
  }

.
.
.
}


/*
 * Fax Gateway Event handler
 */

long GatewayComplete ( long hEvent )
{
 .
 .
 .

 if( sr_getevttype( hEvent ) == TFX_T38GW )
   {
    printf( "Phase E status of gateway operation: %ld\n",
    ATFX_ESTAT( sr_getevtdev( hEvent ) );
   }

 .
 .
 .

}
```

Update the **ATFX_ESTAT( )** function by adding the following Gateway Error values

#define EFX_SIGNALTIMEOUT   214 - Signal timeout - no data or events received during GW session

#define EFX_DCNTIMEOUT   215 - DCN timeout - GW session almost complete but no DCN received

#define EFX_BADIPADDRESS   216 - Bad IP address - T38 subsystem did not get remote IP address - check R4 application

#define EFX_CTBUSERROR   217 - CTBus error w/TDM portion of GW session - check R4 application

3.4.12 Dialogic® IP Media Library API for HMP Programming Guide

There are currently no updates to this document.

3.4.13 Dialogic® IP Media Library API for HMP Library Reference

Update Chapter 2, “Function Information”

Added a new parameter **EVT_TELEPHONY** to the **ipm_EnableEvents( )** function. See Section 1.3, “New Parameter Enables RFC 2833 Reception Event”, on page 19 for more information.

Update to the **ipm_ModifyMedia( )** function

The only scenarios supported are:

Modify From:

- DATA_IP_RECEIVEONLY - receive data from IP network but do not send data
- DATA_IP_SENDONLY - send data to the IP network but do not receive data

Modify To:

- DATA_IP_TDM_BIDIRECTIONAL - full duplex data path between IP network and TDM

Update to Chapter 3, “Events”

For the **IPMEV_QOS_ALARM** event description (PTR# 36741), the IPMEV_QOS_ALARM event description, on page 86, includes the text “No data is returned in the event.” This text is misleading because it implies that it is not possible to determine which alarm triggered the event. The description should read:

IPMEV_QOS_ALARM

Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when desired QoS alarm triggers. See the code example (specifically the **CheckEvent( )** function definition) in Section 8.7, “Example Code for QoS Alarm Handling” of the IP Media Library API for Host Media Processing Programming Guide for information on determining which alarm triggered the event.

A new IPMEV_TELEPHONY_EVENT replaces IPMEV_RFC2833SIGNALRECEIVED, which is being deprecated. See Section 1.3, “New Parameter Enables RFC 2833 Reception Event”, on page 19 for more information.

Update to Chapter 4, “Data Structures”

Update to the IPM_PARM_INFO data structure:
In Table 2, the description for the PARMCH_ECNLP_ACTIVE parameter should be: Enables/disables Non-Linear Processing (NLP) value for echo cancellation. NLP is a process in which signals that have a level below a defined threshold are replaced by comfort noise and signals that have a level above the threshold are passed unmodified.

A new data structure, IPM_TELEPHONY_INFO, is associated with the IPMEV_TELEPHONY_EVENT event instead of the current data structure, IPM_RFC2833_SIGNALID_INFO, which is being deprecated. See Section 1.3, “New Parameter Enables RFC 2833 Reception Event”, on page 19 for more information.

3.4.14 Dialogic® SRL API Programming Guide

There are currently no updates to this document.

3.4.15 Dialogic® SRL API Library Reference

There are currently no updates to this document.

3.4.16 Station Side Interface API Library Reference

There are currently no updates to this document.

3.4.17 Dialogic® Voice API for HMP Programming Guide

There are currently no updates to this document.

3.4.18 Dialogic® Voice API for HMP Library Reference

The following information needs to be added to the Voice API for Host Media Processing Library Reference:

Update to **dx_listenEx( )** function (Defect # IPY00032425)

In the Cautions section, the following caution is added:

- It is recommended that you use **dx_listenEx( )** and **dx_unlistenEx( )** in your application, rather than **dx_listen( )** and **dx_unlisten( ).** In particular, do not use both pairs of functions on the same channel. Doing so may result in unpredictable behavior.

Update to **dx_unlistenEx( ) function** (Defect # IPY00032425)

In the Cautions section, the following caution is added:

- It is recommended that you use **dx_listenEx( )** and **dx_unlistenEx( )** in your application, rather than **dx_listen( )** and **dx_unlisten( ).** In particular, do not use both pairs of functions on the same channel. Doing so may result in unpredictable behavior.

3.5 Demonstration Software Documentation Updates

This section contains updates to the following documents:

- Dialogic® Audio Conferencing API for HMP Demo Guide
- Dialogic® CSP API for HMP Demo Guide
- Dialogic® Global Call API for HMP Demo Guide
- Dialogic® IP Media Server for HMP Demo Guide
- Dialogic® IP Gateway (Global Call) for HMP Demo Guide
- Dialogic® Global Call API for HMP Demo Guide

- Dialogic® IP Media Server for HMP Demo Guide

3.5.1 Dialogic® Audio Conferencing API for HMP Demo Guide

There are currently no updates to this document.

3.5.2 Dialogic® CSP API for HMP Demo Guide

There are currently no updates to this document.

3.5.3 Dialogic® Global Call API for HMP Demo Guide

There are currently no updates to this document.

3.5.4 Dialogic® IP Media Server for HMP Demo Guide

There are currently no updates to this document.

3.5.5 Dialogic® IP Gateway (Global Call) for HMP Demo Guide

There are currently no updates to this document.

3.5.6 Ansrmt Voice Demo Online Help

There currently are no changes to this document.

3.5.7 Xaansr Voice Demo Online Help

The following procedure replaces the existing procedure in the **"Running the Demo"** topic (PTR# 36541):

1. Select Start from the Action menu. The voice channels configured in the Options window are displayed.

2. Dial an extension number configured for one of the voice channels, if available.

3. As the voice prompt plays, write down the 4-digit access code that is displayed next to the voice channel in use. The 4-digit access code takes the form n234, where n is the last digit of the voice channel number.

4. After the voice prompt ends, record a brief message and hang up the telephone.

Note: You can stop the recording at any time by pressing any digit.

5. Redial the same extension number, if applicable, as in step 2.

6. During the voice prompt, enter the 4-digit access code using the telephone keypad. The system plays the previously recorded message.

7. Hang up the telephone.

3.5.8 VoiceDemo Online Help

There currently are no changes to this document.