Dialogic®
Making Innovation Thrive™

Application Note

Migrating Applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software to Dialogic® PowerMedia™ Host Media Processing Software

Migrating Applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software to Dialogic® PowerMedia™ Host Media Processing Software

Application Note

Executive Summary

This application note provides guidelines for migrating existing applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software Release 2.0 for Windows® (Dialogic PacketMedia HMP Software 2.0) to Dialogic® PowerMedia™ Host Media Processing Software (PowerMedia HMP). It provides a brief overview of the PowerMedia HMP APIs used to build applications, shows PowerMedia HMP call flows, and provides tables that map Dialogic® NaturalAccess™ Software (NaturalAccess Software) to PowerMedia HMP functions, events, and call states.

Also, application developers will find two demonstration applications that can be downloaded and used to compare the call flows between NaturalAccess Software and PowerMedia HMP. The demos allow the application developer to place a call, play audio from a file, record audio to a file, and disconnect a call.

Note: Dialogic® Host Media Processing Software has joined the Dialogic® PowerMedia™ Media Processing Product Family and is now known as Dialogic® PowerMedia™ Host Media Processing Software (PowerMedia HMP).

Migrating Applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software to Dialogic® PowerMedia™ Host Media Processing Software

Application Note

Table of Contents

Migrating Applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software to Dialogic® PowerMedia™ Host Media Processing Software

Application Note

Introduction

Application developers wanting to migrate their existing applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software Release 2.0 (Dialogic PacketMedia HMP Software 2.0) to Dialogic® PowerMedia™ Host Media Processing Software (PowerMedia HMP) will find the guidelines, samples, and demos provided in this application note useful.

An understanding of Dialogic APIs is beneficial when migrating, and this application note provides tables that map Dialogic® NaturalAccess™ Software (NaturalAccess Software) to PowerMedia HMP functions, events, and call states. The appropriate documentation is referenced throughout to further support the application developer's efforts.

Other areas discussed include determining which Dialogic® Standard Runtime Library to use, selecting a call control model, and using Dialogic® Global Call API for call processing.

Two demonstration applications (see "For More Information" for the web links to the downloads) are referenced that application developers can run to compare the call flows between NaturalAccess Software and PowerMedia HMP, allowing them to place a call, play audio from a file, record audio to a file, and disconnect a call.

Overview

PowerMedia HMP provides services for building flexible, scalable, and cost-effective IP and 3G-324M multimedia platforms. The software provides these services on general-purpose servers without requiring the use of specialized hardware.

Migrating Applications from Dialogic® NaturalAccess™ PacketMedia™ Host Media Processing Software to Dialogic® PowerMedia™ Host Media Processing Software

Dialogic® PowerMedia™ Host Media Processing Software APIs

PowerMedia HMP applications can be written using the Dialogic APIs described in Table 1. To use one of these APIs as part of an application, include the appropriate library and header file.

Dialogic® APIs	Description	Library/Header Files
Conferencing API	Controls conferencing	Cnflib.h Cnferrs.h Cnfevts.h Libcnf.lib
Continuous Speech Processing API	Allows for the integration of enhanced speech processing algorithms	Eclib.h Libecmt.lib
Device Management API	Allows for the interconnection of devices along the CTBus or Packet Bus	Devmgmt.h Libdevmgmt.lib
Fax API	Controls fax processing	Faxlib.h Libfaxmt.lib
Global Call API	Establishes calls and manages call states Can be run in two modes: First Party Call Control (1PCC) mode and Third Party Call Control (3PCC) mode	gclib.h gcerr.h gcip.h gcip_defs.h gcipmlib.h libgc.lib
IP Media Library API	Controls the RTP stream when running in 3PCC mode or using a non-Dialogic SIP stack	Ipmlib.h Ipmerror.h
Standard Runtime Library (SRL)	Retrieves and manages asynchronous events and timeouts	srllib.h libsrlmt.lib
Voice Library	Processes media, including playing, recording, dialing, and DTMF processing	dxxlib.h libdxxmt.lib

Table 1. Dialogic® APIs for Applications Deployed with Dialogic® PowerMedia™ Host Media Processing Software

Dialogic® PowerMedia™ Host Media Processing Software HMP Documentation

PowerMedia HMP releases have associated sets of manuals, including the following:

Dialogic® Host Media Processing Software Installation Guide — Explains how to install PowerMedia HMP and update an existing software version.

Dialogic® Host Media Processing Software Administration Guide (Windows® only) — Describes how to perform the various tasks related to obtaining and activating Dialogic PowerMedia HMP software license files.

Dialogic® System Configuration Guide — Explains how to configure Dialogic® software, including PowerMedia HMP.

Dialogic® Global Call IP Technology Guide — Provides IP-specific information for the Dialogic® Global Call API.

Dialogic® Host Media Processing API Programming Guides — Provides guidelines for application developers and is useful material to review before starting detailed implementations.

Note: Most APIs supported by PowerMedia HMP have their own API Programming Guide.

Dialogic® Host Media Processing API Library Reference Manuals — Most Dialogic APIs supported by PowerMedia HMP have their own API Library Reference Manual, containing information about the functions, data structures, and events used as part of the API library, and which also shows sample code for the functions.

Dialogic® Host Media Processing Demo Guides — Describes the PowerMedia HMP Software demonstration applications and provides instructions for running the applications on the Linux and Windows® operating systems.

It is useful to become familiar with the API Programming Guides, API Library Reference Manuals, and the *Dialogic® Global Call IP Technology Guide* before planning a migration.

To access PowerMedia HMP documentation:

1. Navigate to the Documentation section of the Dialogic web site (http://www.dialogic.com/manuals).
2. On the left side of the page, click the link for the PowerMedia HMP release desired.
3. In the Programming Libraries section, click the link for the manual desired.

Installing and Configuring Dialogic® PowerMedia™ Host Media Processing Software

To install and configure PowerMedia HMP, including related demonstration programs, do the following:

1. Download the PowerMedia HMP release from the Dialogic website as follows:
   a. Access the "Dialogic® Host Media Processing Software – Evaluating and Licensing" page.
   b. In the Current Releases section, click the link for the version to install.
   c. In the Current Version section, click the link for the listed service update.
2. PowerMedia HMP software comes with a one-port evaluation license, but an option to register and download an additional two-port evaluation license from the "Dialogic Host Media Processing Software – Evaluating and Licensing" page is also available. To obtain additional evaluation licenses, contact your Dialogic Sales representative.
3. Follow the instructions for downloading the software.
4. Install the software, as described in the *Dialogic® Host Media Processing Software Installation Guide*.
5. Run the Dialogic® HMP License Manager to activate the PowerMedia HMP license, as described in the *Dialogic® Host Media Processing Software Administration Guide*.
6. Run the Dialogic® Configuration Manager (DCM) to detect the license and start the HMP services, as described in the *Dialogic® Host Media Processing Software Administration Guide*.
7. Validate the configuration by creating and receiving calls using the PowerMedia HMP based siphmpdemo application. For more information, see "Running the SIP Demonstration Applications" in this application note.

The Dialogic Helpweb has notes on a variety of topics and can be used for troubleshooting or research related to PowerMedia HMP installation, and for other procedures.

Using the Dialogic® Standard Runtime Library for Event Processing

The Dialogic® Standard Runtime Library (SRL) retrieves and processes application events for PowerMedia HMP libraries.

When migrating applications from Dialogic PacketMedia HMP Software 2.0 to PowerMedia HMP, first decide which SRL programming model to use. As of the publication of this application note, PowerMedia HMP supports the following programming models:

- Single threaded polled model (asynchronous)
- Multithreaded polled model (asynchronous)
- Callback model (asynchronous)
- Multithreaded synchronous model

For more information about these programming models, see the *Dialogic® Standard Runtime Library API Programming Guide*.

Selecting one of the polled models can provide a better match for call control applications, and are also more similar to the NaturalAccess Software runtime model than the other SRL programming models.

In the polled models, the sr_waitevt() function works like the NaturalAccess Software ctaWaitEvent() function, waiting for and retrieving events and timeouts. The application can process these received events upon receipt. The SRL creates the needed event queues at application startup, removing the need to call a function like ctaCreateQueue(). However, open the device with the library-specific open function in order to obtain the device handle.

During the migration process, use SRL sr_putevt() to place a user-defined event onto the SRL event queue. This can be helpful for generating events that map into the existing architecture.

Table 2 shows the function mapping between NaturalAccess Software APIs and the SRL API.

NaturalAccess Software API Functions	Standard Runtime Library API Functions
ctaCreateQueue	Not Applicable
ctaWaitEvent	sr_waitevt
Not Applicable	sr_putevt

Table 2. Function Mapping between the Dialogic® NaturalAccess™ Software APIs and the Dialogic® Standard Runtime Library API

For more information about the SRL, refer to the *Dialogic® Standard Runtime Library API Programming Guide* and the *Dialogic® Standard Runtime Library API Library Reference*.

Selecting a Call Control Model

Use one of the following models for implementing IP call control with PowerMedia HMP:

- Dialogic® SIP stack controlled by the Dialogic® Global Call API in 1PCC operating mode
- Dialogic SIP stack controlled by the Global Call API in 3PCC operating mode
- Non-Dialogic SIP stack

Each of these call control models has benefits, but the Global Call API abstracts protocol-specific functions, meaning that an application created with this API does not have to manually walk through the protocol, and instead, just handles call states.

The Global Call API is similar to the Dialogic® NaturalAccess™ SIP for NaturalCallControl™ API (NaturalAccess SIP for NCC API) in that both are high-level APIs, and both have similar machine states and event schemes. However, an application using the Global Call API in 3PCC mode is responsible for creating the Session Description Protocol (SDP) and manually attaching it to the SIP messages. (The SDP is automatically created and attached to SIP messages in the NaturalAccess SIP for NCC API and the Global Call API in 1PCC operating mode.)

Figure 1 shows an example of a typical function, event, and SIP message sequence for a call running in Global Call API 3PCC mode using the Dialogic® IP Media Library API for media control.

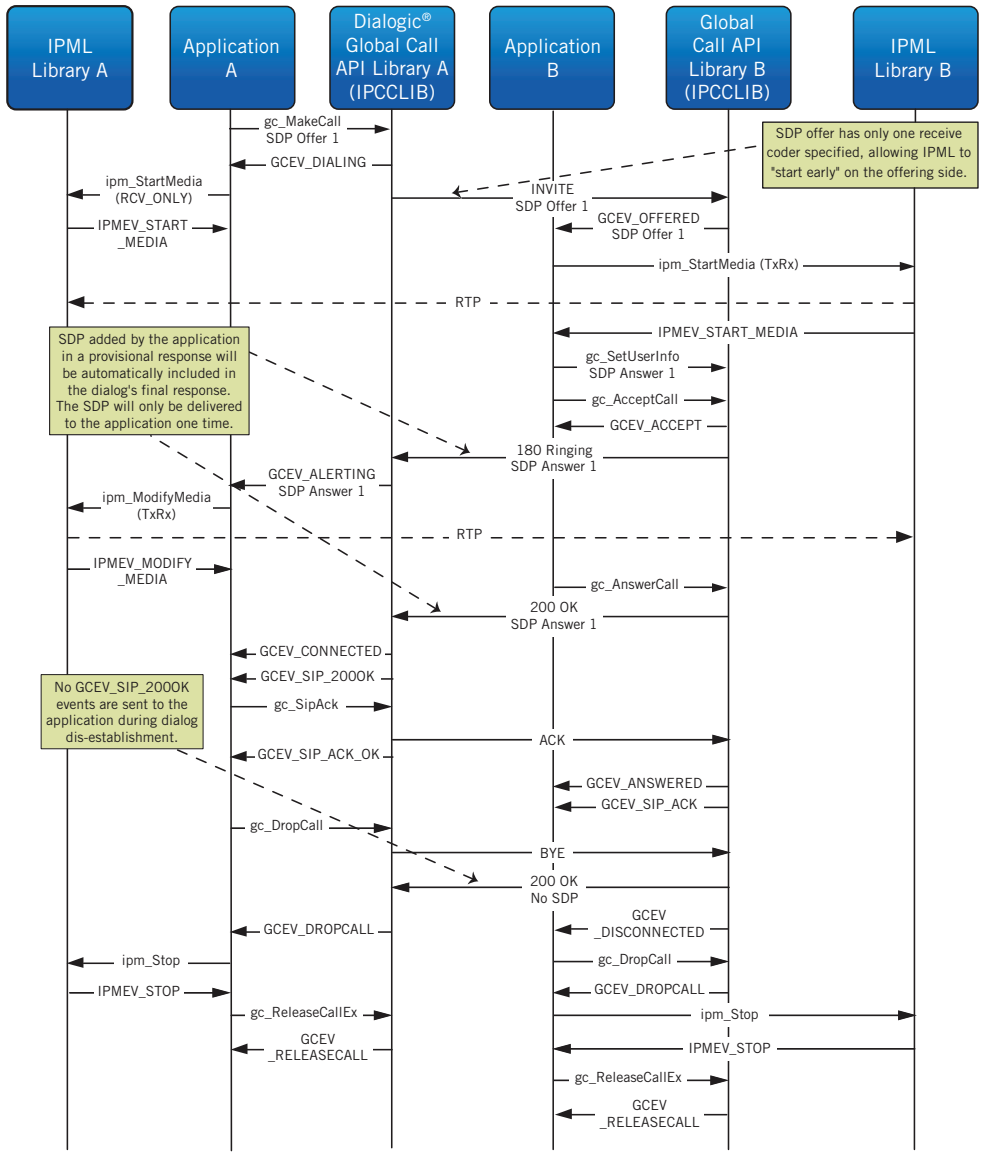


Figure 1. Typical Call Flow for a Call Running in Dialogic® Global Call API 3PCC Operating Mode

Using the Dialogic® Global Call API for Call Processing

This section describes how to use the Global Call API for establishing and controlling calls. The Global Call API implementation is similar for both the 1PCC and 3PCC operating modes. The main difference lies with negotiating media and controlling RTP streaming.

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

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

Initializing the Dialogic® Global Call API Library

Before using Global Call API functions, the application must start the Global Call API library. To start the Global Call API library, initialize Global Call_START_STRUCT and call gc_Start(), as described in the *Dialogic® Global Call API Programming Guide*. gc_Start() maps to ctaCreateQueue() in the NaturalAccess Software API.

The configuration created by gc_Start() goes inside the IP_VIRTBOARD structure, which is the structure that initializes the Dialogic SIP stack. The fields in IP_VIRTBOARD control configuration and capability information, such as the SIP signaling port, number of initialized channels, and the maximum number of Internet Protocol Telephony (IPT) devices that can be used for SIP calls. IP_VIRTBOARD also can be set to specify whether it is running in 1PCC or 3PCC mode and whether the application has access to SIP message information.

For more information, see the *Dialogic® Global Call IP Technology Guide*.

The following example shows how to initialize an IP_VIRTBOARD structure. This example configures the virtual board to run on the default SIP signaling port (5060) and specifies 3PCC as the operating mode:

```
int startGlobalCall(int a_maxCalls)
{
char                              str[MAX_STRING_SIZE];

          GC_START_STRUCT gclib_start;
          IPCCLIB_START_DATA cclibStartData;
          IP_VIRTBOARD virtBoards[1];

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

          INIT_IP_VIRTBOARD(virtBoards);

          virtBoards[0].total_max_calls = a_maxCalls;
          virtBoards[0].h323_max_calls = IP_CFG_NO_CALLS;
          virtBoards[0].sip_max_calls = a_maxCalls;
          virtBoards[0].sip_signaling_port = 5060;
          virtBoards[0].sup_serv_mask = IP_SUP_SERV_CALL_XFER;
          virtBoards[0].sip_msginfo_mask = IP_SIP_MSGINFO_ENABLE|IP_SIP_
          MIME_ENABLE;

          INIT_IPCCLIB_START_DATA(&cclibStartData, 1, virtBoards);
//3PCC
          if(m_3pccMode)
```

```
        {
              cclibStartData.max _ parm _ data _ size = IP _ CFG _ PARM _ DATA _
              MAXLEN;
              cclibStartData.media _ operational _ mode = MEDIA _ OPERATIONAL _
              MODE _ 3PCC;
        }

        CCLIB _ START _ STRUCT cclib _ start[]={
                                    {"GC _ H3R _ LIB", &cclibStartData},
                                    {"GC _ IPM _ LIB", NULL},
                                    {"GC _ DM3CC _ LIB", NULL}};
        gclib _ start.num _ cclibs = 3;
        gclib _ start.cclib _ list = cclib _ start;
        if(gc _ Start(&gclib _ start) == -1)
        {
              sprintf(str, "ERROR in GCSTART");
              printandlog(ALL _ DEVICES, GC _ APIERR, NULL, str, 0);
                    return -1;
        }
    printandlog(ALL _ DEVICES, GC _ APICALL, NULL, "GC _ START SUCCESSFULL", 0);
    return 0;
}
```

Opening a Dialogic® Global Call API Device

The Global Call API opens a device that is a collection of other devices. For example, the opened device might include a voice component for playing and recording, an IPT device used for IP call control, and an IP media device used for IP media RTP streaming. All of these devices combine to form one logical cluster or "channel."

Use gc_OpenEx() to open a Global Call API device and obtain the line device handle. Because Global Call API supports many different protocols, it is useful to specify the protocol to run on this channel. This information is passed to the function via the Open string.

Note: To obtain a handle to an underlying component of a Global Call API device, use gc_GetResourceH().

The following example shows a typical open string for a 1PCC implementation.

":P_SIP:N_iptB1T1:M_ipmB1C1:V_dxxxB1C1"

This string opens call control channel 1 (iptB1T1), media device channel 1 (ipmB1C1), and voice channel 1 (dxxxB1C1), and it associates these channels with the same cluster.

For a 3PCC implementation, the protocol and the network device (":P_SIP:N_iptB1T1") only are commonly used. The media device opens separately via ipm_Open().

gc_Open() takes a user context pointer, which is included with an event delivered to the application. It is useful for storing pointers to classes or application-specific data to speed up the processing of events later on. After calling gc_Open(), the application should wait for a GCEV_OPEN event and a GCEV_UNBLOCKED event before proceeding.

The following example shows an open event that supports both 1PCC and 3PCC operating modes:

```
static void gc_demo_open_SIP_channel(int index)
{
    char str[MAX_STRING_SIZE];
    char ipmname[15];

    printandlog(index, MISC, NULL, "SIP device being opened", 0);

    // dev name set elseware but can be set here via the index
    if(g_3pccMode){
        sprintf(port[index].devname, "P_SIP:N_iptB1T%d",index);
    }
    else{
        sprintf(port[index].devname, "P_SIP:N_iptB1T%d:M_ipmB1C%d",index,
        index);
    }
    if (gc_OpenEx(&port[index].ldev, port[index].devname, EV_ASYNC, (void
        *)&port[index]) != GC_SUCCESS) {
        sprintf(str, "gc_OpenEx(devicename=%s, mode=EV_ASYNC) Failed",
        port[index].devname);
        printandlog(index, GC_APIERR, NULL, str, 0);
        exitdemo(1);
    }

    sprintf(str, "gc_OpenEx(devicename=%s, mode=EV_ASYNC) Success",
    port[index].devname);
    printandlog(index, GC_APICALL, NULL, str, 0);

    if(g_3pccMode){
        printandlog(index, MISC, NULL, "IPM device being opened", 0);

        sprintf(ipmname,"%s",port[index].medianame);

        if ((port[index].mediah=ipm_Open(ipmname, NULL ,EV_ASYNC) )== -1) {
            sprintf(str, "ipm_open(devicename=%s) Failed", ipmname);
            printandlog(index, GC_APIERR, NULL, str, 0);
            exitdemo(1);
        }

        sprintf(str, "ipm_Open(devicename=%s) Success, mediah=%d",
        ipmname,port[index].mediah);
        printandlog(index, GC_APICALL, NULL, str, 0);
    }
}
```

Receiving an Inbound Call

To receive an inbound call with the Global Call API, use gc_WaitCall() to place the channel in a waitcall state. The application will receive a GCEV_OFFERED event when an inbound call arrives. gc_WaitCall() maps to nccStartProtocol() in the NaturalAccess SIP for NCC API.

Figure 2 shows a typical function, event, and SIP message sequence for receiving an inbound call with the Global Call API.

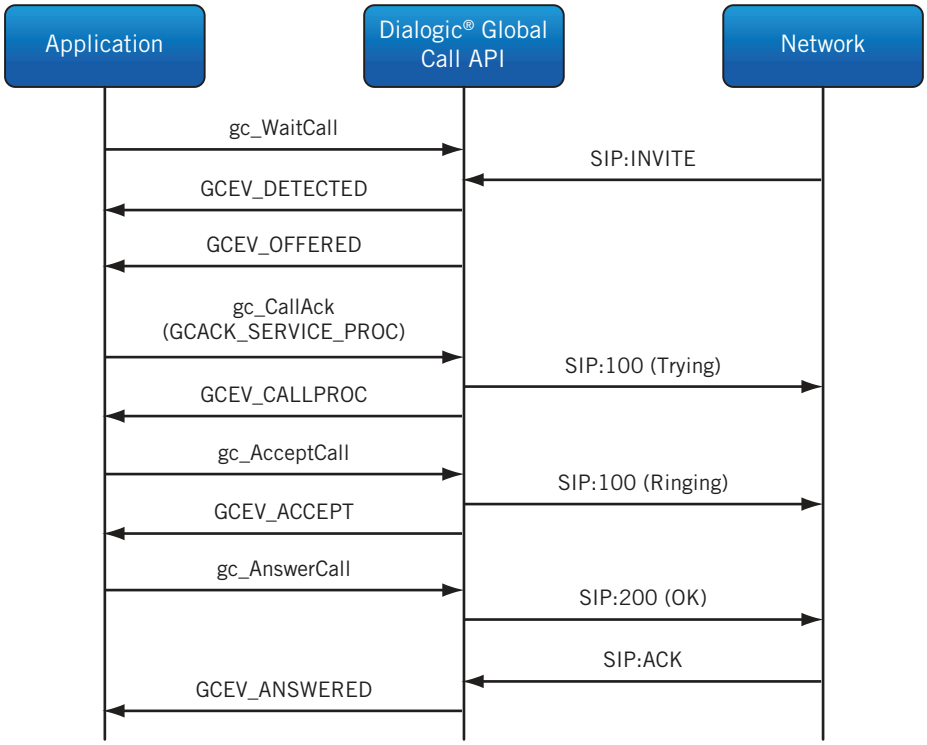


Figure 2. Receiving an Inbound Call with the Dialgoic® Global Call API

For more information about receiving an inbound call with the Global Call API, see the *Dialogic® Global Call IP Technology Guide* and *Dialogic® Global Call Programming Guide*.

Placing an Outbound Call

To place an outbound call with the Global Call API, use gc_MakeCall(). This function maps to nccPlaceCall() in the NaturalAccess SIP for NCC API.

Figure 3 shows a typical function, event, and SIP message sequence for placing an outbound call with the Global Call API.

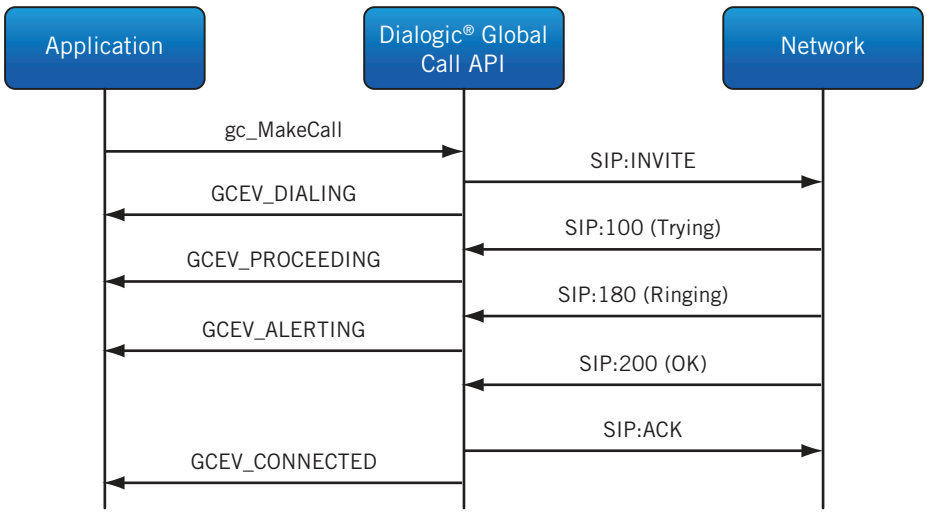


Figure 3. Placing an Outbound Call with the Dialogic® Global Call API

For more information about placing an outbound call with the Global Call API, see the *Dialogic® Global Call IP Technology Guide* and the *Dialogic® Global Call API Programming Guide*.

Tearing Down a Call

To tear down a call with the Global Call API, use gc_DropCall() and gc_Releasecall(). gc_DropCall() maps to nccDisconnectCall() in the NaturalAccess SIP for NCC API, and gc_Releasecall() maps to nccReleaseCall().

Figure 4 shows a typical function, event, and SIP message sequence for a call teardown initiated by the application.

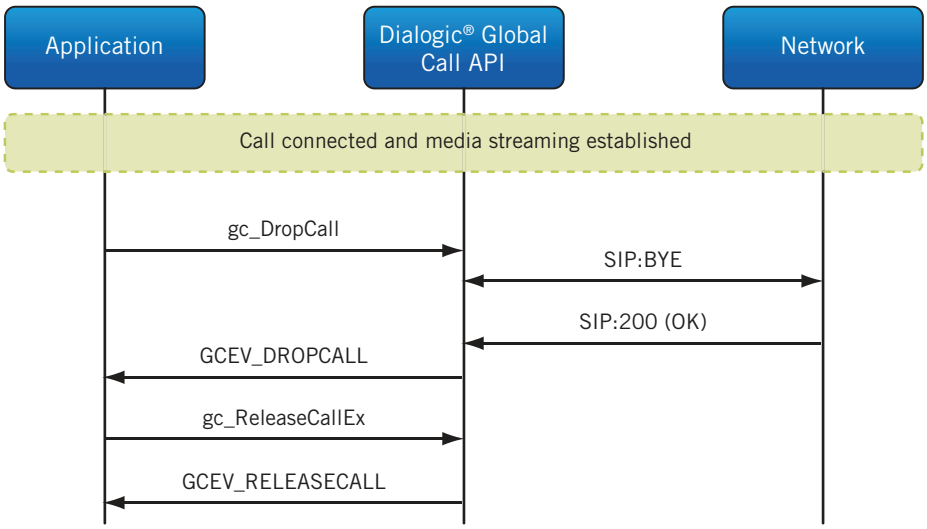


Figure 4. Typical Call Teardown Initiated by the Application

Figure 5 shows a typical function, event, and SIP message sequence for a call teardown initiated by the remote end.

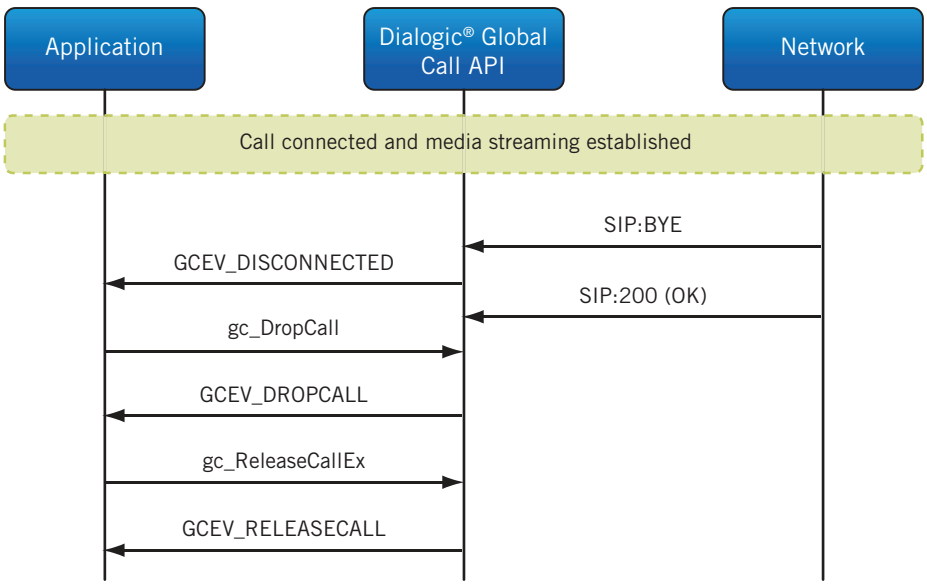


Figure 5. Typical Call Teardown Initiated by the Remote End

For more information about tearing down a call with the Global Call API, see the *Dialogic® Global Call IP Technology Guide* and *Dialogic® Global Call API Programing Guide*.

Managing SIP Message Content

It is often desirable to modify the SIP packet contents in order to interface with devices on the SIP network. Doing so includes changing the default SIP headers and/or changing the SDP information.

In both the 1PCC and 3PCC operating modes, the Global Call API library uses default values for each of the headers. These values are stored inside the Global Call API library and need to be modified using gc_util_insert_parm_ref_ex. For more information, see the *Dialogic® Global Call IP Technology Guide*.

The following example shows how to override the default CONTACT field in the SIP header and set it to Global CallBCM with the local number:

```
int SetContact(struct channel *pline,long l_crn, char *l_contact = NULL){
    char                                str[MAX_STRING_SIZE];
    char contact[120] ;
    char *contact1;
    GC_PARM_BLKP pParmBlock = NULL;

    if(l_contact == NULL){
          sprintf(contact,"Contact: \"GCBCM\" <%s>",pline->destination_num);
          contact1=contact;
    } else{
```

```
            contact1 = l _ contact;
      }

      sprintf(str,"Setting contact to -> %s\n",contact1);
      printandlog(pline->index, MISC, NULL, str, 0);


      gc _ util _ insert _ parm _ ref _ ex(&pParmBlock,
                  IPSET _ SIP _ MSGINFO,
                  IPPARM _ SIP _ HDR,
                  (unsigned long) (strlen(contact1)+1),
                  contact1);

      if(pParmBlock == NULL){
            sprintf(str,"Error allocating pParmBlock from
            gc _ util _ insert _ parm _ ref _ ex\n");
            printandlog(pline->index, GC _ APIERR, NULL, str, 0);
            return GC _ ERROR;
      }
      int frc = gc _ SetUserInfo(GCTGT _ GCLIB _ CRN,l _ crn,pParmBlock,GC _
      SINGLECALL);
      if(GC _ SUCCESS != frc)
      {
            sprintf(str,"Error setting gc _ SetUserInfo\n");
            printandlog(pline->index, GC _ APIERR, NULL, str, 0);
            gc _ util _ delete _ parm _ blk(pParmBlock);
            return GC _ ERROR;
      }
            gc _ util _ delete _ parm _ blk(pParmBlock);
            return GC _ SUCCESS;
}
```

Controlling SIP Media Negotiation

When running in 1PCC operating mode, the SDP is automatically inserted inside the appropriate SIP messages. Use gc_util_insert_parm_val to insert parameters that enable coder support, RFC 2833 enablement, and other functionality. For more information, see the *Dialogic® Global Call IP Technology Guide*.

The following example enables RFC 2833 digit negotiation.

```
//3pcc
// In 3pcc mode this is done manually
if(!g _ 3pccMode){
      gc _ util _ insert _ parm _ val(&parmblkp, IPSET _ DTMF, IPPARM _ SUPPORT _
      DTMF _ BITMASK,
            sizeof(char), IP _ DTMF _ TYPE _ RFC _ 2833);
```

```
    if (gc_SetUserInfo(GCTGT_GCLIB_CHAN, port[index].ldev, parmblkp, GC_
    ALLCALLS) !=
          GC_SUCCESS) {
          sprintf(str, "gc_SetUserInfo(linedev=%ld) Failed configuring DTMF
          mode",
          port[index].ldev);
          printandlog(index, GC_APIERR, NULL, str, 0);
          exitdemo(1);
    }
sprintf(str, "gc_SetUserInfo(linedev=%ld) Success - DTMF mode is RFC2833",
port[index].ldev);
printandlog(index, GC_APICALL, NULL, str, 0);
gc_util_delete_parm_blk(parmblkp);
}
```

When running in the 3PCC operating mode, the SDP is not generated by the Global Call API library and must be manually built by the application. Sample code for building the SDP is available with the PowerMedia HMP download (available in the SDPAPI demo in the \dialogic\demos\sdpapi directory).

The following example uses the SDP library to generate the SDP that will be delivered in response to a SIP INVITE:

```
.
.
. GCEV_OFFERED Event processing here

//3pcc
if(g_3pccMode){
    // Use the sdpapi to build an SDP, using the media types, ports
    // and IP address extracted from the SDP that came with the INVITE
    offerSDP.clear();
    BuildOfferSDP(&offerSDP, pline->index);
    if (offerSDP.exportSDP(sdp, 1024, false) == 0) {
                printandlog(index, MISC, NULL, "Export of Offer SDP
                successful\n",
                0);
    } else {
          printandlog(index, MISC, NULL, "ERROR building Offer SDP \n", 0);
    }
    // Add 1 to strlen for null termination
    len = strlen(sdp) + 1;

    // Note that the "ex" version is used here to handle potentially long
       parameter (SDP)
       if (gc_util_insert_parm_ref_ex(&parmblkp, IPSET_SDP,IPPARM_SDP_
       OFFER, len,
                sdp) == -1)
    {
```

```
            sprintf(str, "gc _ util _ insert _ parm _ ref _ ex() Failed", pline->ldev,
            pline-
            >destination _ num);
            printandlog(index, GC _ APIERR, NULL, str, 0);
            }

      if(gc _ SetUserInfo(GCTGT _ GCLIB _ CHAN, pline->ldev,parmblkp , GC _ NEXT _
      OUTBOUND _ MSG)
            == -1)
      {
            sprintf(str, "gc _ SetUserInfo(SDP) Failed", pline->call[callindex].crn);
            printandlog(index, GC _ APIERR, NULL, str, 0);
      }
}
if (gc _ AnswerCall(pline->call[callindex].crn, 0, EV _ ASYNC) != GC _ SUCCESS) {
            sprintf(str, "gc _ AnswerCall(crn=0x%lx, # of rings=0, mode=EV _ ASYNC)
            Failed",
            pline->call[callindex].crn);
            printandlog(index, GC _ APIERR, NULL, str, 0);
            exitdemo(1);
}
sprintf(str, "gc _ AnswerCall(crn=0x%lx, mode=EV _ ASYNC) Success", pline-
>call[callindex].crn);
printandlog(index, GC _ APICALL, NULL, str, 0);
.
.
.
```

Where the BuildOfferSDP has the following syntax:

```
void BuildOfferSDP(sdpSessionDescription *sdp, int index)
{

      sdp->version()->setVersion("0");

      sdp->origin()->setUserName("3PCC _ GCBC");
      sdp->origin()->setSessionId("1234");
      sdp->origin()->setVersion("5678");
      sdp->origin()->setNetworkType("IN");
      sdp->origin()->setAddressType("IP4");
      sdp->origin()->setAddress(port[index].LocalIpAddr);

      sdp->sessionName()->setName("Dialogic _ SIP _ CCLIB");
      sdp->sessionInformation()->setInfo("3PCC _ GCBC Session Info");

      sdp->connection()->setNetworkType("IN");
      sdp->connection()->setAddressType("IP4");
```

```
    sdp->connection()->setAddress(port[index].LocalIpAddr);

    sdpTimeDescription* pTD=sdp->timeDescriptionList()->addItem();
    unsigned long Start = 0;
    unsigned long Stop = 0;
    pTD->time()->setStart(Start);
    pTD->time()->setStop(Stop);

    sdpAttribute* pAttribute = sdp->attributeList()->addItem();
    sdpMediaDescription* pMD=sdp->mediaDescriptionList()->addItem();

    pMD->media()->setMedia("audio");
    pMD->media()->setPort(49150 + (index*2));
    pMD->media()->setTransport("RTP/AVP");
    pMD->media()->setNumPorts(1);

    // Need to parse the coder list and add here
    for(int i=0;i<port[index].num _ codecs;i++){
          pMD->media()->addFormat(codeToFormat(port[index].ipcap[i].capability));
}
// If RFC 2833
    pMD->media()->addFormat("101");

    sdpAttributeList* pAttributeList = pMD->attributeList();
          for(i=0;i<port[index].num _ codecs;i++){
                setMediaAttributes(&port[index].ipcap[i], pAttributeList);
          }

          //If RFC2833
    pAttribute = pAttributeList->addItem();
    pAttribute->setProperty("rtpmap");
    pAttribute->setPropertyValue("101 telephone-event/8000");
}
```

Controlling the Media Stream

The RTP media stream is started after a call is connected as follows:

- In the 1PCC operating mode, the RTP media stream is started automatically by the Global Call API Libraries.
- In the 3PCC operating mode, the application uses the IP Media Library API to start the media stream. This API uses the device handle obtained by the ipm_Open, instead of the Global Call API device handle. It is desirable to parse the SIP messages to see which of the SDP members was successfully negotiated.

For more information about the IP Media Library API, see the *Dialogic® IP Media Library API Programming and Library Reference*.

Dialogic® NaturalAccess™ Software API to Dialogic® Global Call API Mappings

This section maps NaturalAccess Software API functions, events, and call states to the corresponding Global Call API functions, events, and call states.

Function Mapping

Table 3 shows the mapping between NaturalAccess Software API functions and Global Call API functions.

NaturalAccess Software API Software API Function	Global Call API Function
ctaCloseServices	gc_Detatch
ctaCreateContext, ctaOpenServices, and nccStartProtocol, in sequence	gc_Open or gc_OpenEx
ctaCreateQueue	gc_Start
ctaDestroyQueue	gc_Stop
ctaGetContextInfo	gc_GetUsrAttr
ctaGetParmByName	gc_GetParm
ctaGetText	gc_ErrorInfo
ctaGetTextEx	gc_ErrorValue
ctaInitialize followed by ctaCreateQueue	gc_Start
ctaOpenServices	gc_Attach or gc_AttachResource
ctaSetParmByName	gc_SetParm
ctaStartTrace	gc_StartTrace
ctaStopTrace	gc_StopTrace
ctaWaitEvent	gc_GetMetaEvent or gc_GetMetaEventEx
nccAcceptCall	gc_CallAck or gc_AcceptCall, depending on the protocol
nccAnswerCall	gc_CallAck or gc_AnswerCall, depending on the protocol
nccDisconnectCall	gc_DropCall
nccGetCallStatus	gc_CRN2LineDev, gc_GetANI, gc_GetCallState, or gc_GetDNIS
nccGetExtendedCallStatus	gc_GetSigInfo
nccGetLineStatus	gc_GetLineDevState or gc_GetXmitSlot
nccPlaceCall	gc_MakeCall
nccRejectCall	gc_CallAck
nccReleaseCall	gc_DropCall, gc_ReleaseCall, or gc_ReleaseCallEx
nccStartProtocol	gc_WaitCall
nccStopProtocol	gc_Close
nccStopProtocol, ctaCloseServices, and ctaDestroyContext, in sequence	gc_Close
nccStopProtocol followed by nccStartProtocol,	gc_ResetLineDev
no-op (if an internal timeout is reached)	gc_CallAck
oamBoardGetProduct followed by oamGetKeyword	gc_GetCTInfo
swiDisableOutput	gc_UnListen
swiMakeConnection	gc_Listen

Table 3. Mapping of Dialogic® NaturalAccess™ Software API Functions to Dialogic® Global Call API Functions

Event Mapping

Table 4 shows the mapping between NaturalAccess Software API events and Global Call API events.

NaturalAccess Software API Events	Global Call API Events
ADIEVN_PROTOCOL_ERROR	GCDEV_ERROR or GCEV_TASKFAIL
CTAEVN_CLOSE_SERVICES_DONE with the value field equal to CTA_REASON_FINISHED	GCEV_DETACH
CTAEVN_CLOSE_SERVICES_DONE with the value field different from CTA_REASON_FINISHED, indicating the reason of the failure	GCEV_DETACH_FAIL
CTAEVN_OPEN_SERVICES_DONE with the value field equal to CTA_REASON_FINISHED	GCEV_OPENEX followed by GCEV_ATTACH
CTAEVN_OPEN_SERVICES_DONE with the value field different from CTA_REASON_FINISHED	GCEV_OPENEX followed by GCEV_OPENEXFAIL
NCCEVN_ACCEPTING_CALL	GCEV_ACCEPT
NCCEVN_ANSWERING_CALL and NCCEVN_CALL_CONNECTED	GCEV_ANSWERED
NCCEVN_CALL_DISCONNECTED with reason code in the value field set to NCC_DIS_TIMEOUT or NCC_DIS_REMOTE_NOANSWER	GCEV_DISCONNECTED and GCEV_CALLSTATUS
NCCEVN_CALL_RELEASED	GCEV_RELEASECALL
NCCEVN_EXT_CALL_STATUS_UPDATE with the value field set to CALL_STATUS_UUI	GCEV_USRINFO
NCCEVN_PLACING_CALL	GCEV_SETUP_ACK
NCCEVN_PLACING_CALL followed by NCCEVN_CALL_PROCEEDING	GCEV_PROCEEDING
NCCEVN_PROTOCOL_ERROR	GCEV_ERROR or GCEV_TASKFAIL
NCCEVN_PROTOCOL_EVENT with the value field ISDN_PROGRESS	GCEV_PROGRESSING
NCCEVN_REMOTE_ALERTING	GCEV_ALERTING
NCCEVN_REMOTE_ANSWERED and NCEVN_CALL_CONNECTED, in sequence	GCEV_CONNECTED
NCCEVN_CALL_DISCONNECTED	GCEV_DROPCALL
NCCEVN_CALL_RELEASED	GCEV_DROPCALL
NCCEVN_SEIZURE_DETECTED followed by NCCEVN_INCOMING_CALL	GCEV_OFFERED
NCCEVN_START_PROTOCOL_DONE with the value field CTA_REASON_FINISHED	GCEV_RESETLINEDEV
NCCEVN_START_PROTOCOL_DONE with the value field different from CTA_REASON_FINISHED	GCEV_RESTARTFAIL
NCCEVN_STOP_PROTOCOL_DONE	GCEV_RESETLINEDEV

Table 4. Mapping of Dialogic® NaturalAccess™ Software API Events to Dialogic® Global Call API Events

Call State Mapping

Table 5 shows the mapping between NaturalAccess Software API call states and Global Call API call states.

NaturalAccess Software API Call State	Global Call API Call State
NCC_CALLSTATE_ACCEPTING	GCST_ACCEPT
NCC_CALLSTATE_PROCEEDING	GCST_ALERTING
NCC_CALLSTATE_CONNECTED	GCST_CONNECTED
NCC_CALLSTATE_SEIZURE	GCST_DETECTED
NCC_CALLSTATE_OUTBOUND_INITIATED	GCST_DIALING
NCC_CALLSTATE_DISCONNECTED	GCST_DISCONNECTED
NCC_CALLSTATE_RECEIVING_DIGITS	GCST_GETMOREINFO
NCC_CALLSTATE_DISCONNECTED	GCST_IDLE
NCC_CALLSTATE_INVALID	GCST_NULL
NCC_CALLSTATE_INCOMING	GCST_OFFERED
NCC_CALLSTATE_PROCEEDING	GCST_PROCEEDING
NCC_CALLSTATE_PLACING	GCST_SENDMOREINFO

Table 5. Mapping of Dialogic® NaturalAccess™ Software Call States to Dialogic® Global Call API Call States

Running the SIP Demonstration Applications

Dialogic provides two demonstration applications (see "For More Information") that can be used to compare the call flows between NaturalAccess Software and PowerMedia HMP. Both demos are called siphmpdemo, and they allow application developers to perform the following operations:

- Place a call
- Play audio from a file
- Record audio to a file
- Disconnect a call

The PowerMedia HMP based siphmpdemo application also allows for configuring PowerMedia HMP options, such as selecting voice media and vocoder codecs.

Running the Dialogic® NaturalAccess™ Software Based siphmpdemo Application

To run the NaturalAccess Software based siphmpdemo application (available for download; see "For More Information"), do the following:

1. Extract files from NA-SipHMPdemo.zip. By default, the files will be extracted to \dialogic\packetmedia\siphmpdemo.
2. Install Dialogic® NaturalAccess™ Software (R8.0 or later).
3. Install SIP 2.0.
4. Install a SIP evaluation license, if no SIP license is currently installed.

5. Start ctdaemon, if it is not already started as service.

6. Start the SIP server, if it is not already started as service.

7. Run oamsys from the directory that contains the files from NA-SipHMPdemo.zip.

Optionally, open the siphmpdemo project in Microsoft® Visual Studio® and build the demo (for convenience, the demo file includes a pre-built executable [siphmpdemo.exe]), as follows:

1. In a command window, launch siphmpdemo with the –h option to view the available configuration options.

```
>siphmpdemo –h
```

siphmpdemo displays the following text:

```
Usage: siphmpdemo [Options]

    PROGRAM SETUP:
    -b HMP Board Number                          (Default 0)
    -d enable debug logging                      (Default 0)
    -D RFC 2833 payload ID                       (Default 0, using inband DTMFs)
    -s Timeslot                                  (Default 0)

    IP OPTIONS:
    -L Local IP (x.y.z.w)                        (Default 127.0.0.1)
    -l Local UDP Port Number                     (Default same as remote port)

    VOICE MEDIA OPTIONS:
    -e Record Encoding                           (Default 10)
          Supported Formats include:
                1 - 16 bit NMS ADPCM
                2 - 24 bit NMS ADPCM
                3 - 32 bit NMS ADPCM
                10 - Mu-Law
                11 - A-Law
                13 - 16-bit PCM 8kbit/s
                15 - 32 bit OKI ADPCM
                20 - 32 bit G.726 ADPCM
                23 - 32 bit IMA ADPCM
                28 - G.729A

VOCODER OPTION :
    -v Channel code                              (Default 0)
          Channel codes include:
                0 - G711 Mu-law
                1 - G729a/b
                2 = G711 A-law
                4 = G726 32kbps
```

2. Configure siphmpdemo, as needed, using the options shown above.

3. In a command window, launch siphmpdemo with the –L option, and enter the local IP address of the server where PowerMedia HMP is installed. For example:

```
siphmpdemo –L 10.128.18.102
```

siphmpdemo displays the following text:

```
*-----------------------------------------------------*
|    SIP/HMP Demo  (NMS flavor)                       |
*-----------------------------------------------------*

Using inband DTMFs
CT Access Environment Initialized
SIP protocol started

                Type 'c number' Place call.
                Type 'd' Disconnect call.
                Type 'p filename' Play from file.
                Type 'r filename' Record to a file.
                Type 'q' to quit.
```

4. Use siphmpdemo to place a call, play and/or record to a file, and disconnect a call.

5. Type q to quit the application.

Running the Dialogic® PowerMedia™ Host Media Processing Software Based siphmpdemo Application

To perform the operations in this section, PowerMedia HMP should be installed as referenced in the "Installing and Configuring PowerMedia HMP" section of this application note.

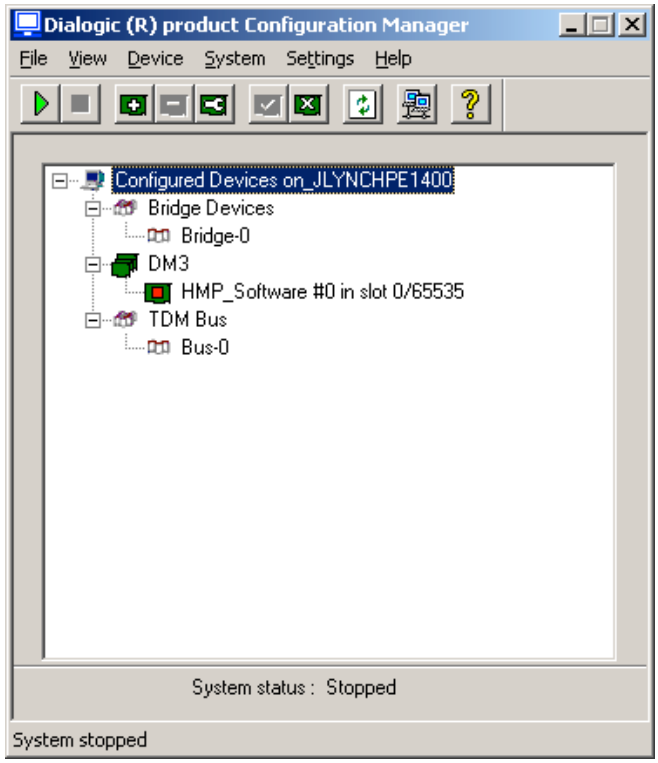
To run the PowerMedia HMP based siphmpdemo application (available for download; see "For More Information"), do the following:

1. Extract the siphmpdemo.zip file. The files will be extracted by default to \dialogic\hmp\siphmpdemo.

2. Optionally, open the siphmpdemo project in Microsoft® Visual Studio® and build the demo. For convenience, the demo file includes a pre-built executable (siphmpdemo.exe).

3. Start the Dialogic Configuration Manager (DCM) from the task bar (Start>Programs>Dialogic HMP>Configuration Manager - DCM).

A window similar to the following displays:



4. Click the green arrow (>) in the toolbar near the top of the window to start PowerMedia HMP. The system might take a few minutes to start.

5. In a command window, Launch siphmpdemo with the –h option to view the available configuration options.

```
>siphmpdemo –h
```

siphmpdemo displays the following text:

```
Usage: siphmpdemo [Options]
     PROGRAM SETUP:
     -b HMP Board Number                          (Default 0)
     -D RFC 2833 payload ID                       (Default 0, using inband DTMFs)
     -s Timeslot                                  (Default 0)

     IP OPTIONS:
     -L Local IP (x.y.z.w)                        (Default 127.0.0.1)
     -l Local UDP Port Number                     (Default same as remote port
     VOICE MEDIA OPTIONS:
     -e Play/Record Encoding                      (Default 7)
           Supported Formats include:
                 1 - 16 bit NMS ADPCM
```

```
                2 - 24 bit NMS ADPCM
                3 - 32 bit NMS ADPCM
                10 - Mu-Law
                11 - A-Law
                13 - 16-bit PCM 8kbit/s
                15 - 32 bit OKI ADPCM
                20 - 32 bit G.726 ADPCM
                23 - 32 bit IMA ADPCM
                28 - G.729A

    VOCODER OPTION :
    -v Channel code                              (Default 0)
          Channel codes include:
                0 - G711 Mu-law
                1 - G729a/b
                2 = G711 A-law
                4 = G726 32kbps
```

6. Configure siphmpdemo, as needed, using the options shown above.

7. In a command window, launch siphmpdemo with the –L option, and enter the local IP address of the server where PowerMedia HMP is installed. For example:

```
    siphmpdemo –L 10.128.18.102
```

siphmpdemo displays the following text:

```
Using inband DTMFs
*-----------------------------------------------------*
|    SIP/HMP Demo (Dialogic flavor)                   |
*-----------------------------------------------------*

Using inband DTMFs
SRL Model Set to SR_STASYNC | SR_POLLMODE
SIP device being opened
Opened :N_iptB1T1:P_SIP:M_ipmB1C1
Timeslot 0: Voice TS = 4096

                Type 'c number' Place call.
                Type 'd' Disconnect call.
                Type 'p filename' Play from file.
                Type 'r filename' Record to a file.
                Type 'q' to quit.
```

8. Use siphmpdemo to place a call, play and/or record to a file, and disconnect a call.

9. Type q to quit the application.

For More Information

Zip files containing the demonstration files can be downloaded:

Dialogic-SipHMPdemo

NA-SipHMPdemo

A Readme file, “siphmpdemo.doc”, for the sample code “siphmpdemo.doc” is available with the demos.

Dialogic®
Making Innovation Thrive™

www.dialogic.com

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9

08/10 11875-01