Dialogic®

Dialogic® Conferencing API

Programming Guide

August 2007

05-2505-002

Publication Date: August 2007

Document Number: 05-2505-002

Contents

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2505-002	August 2007	Made global changes to reflect Dialogic brand.
05-2505-001	August 2006	Initial version of document.

About This Publication

The following topics provide more information about this publication:

Purpose

This publication provides programming guidelines for the Dialogic® Conferencing (CNF) API, supported in Dialogic® Host Media Processing (HMP) Software for Linux and Windows® operating systems. It is a companion document to the *Dialogic® Conferencing API Library Reference*, which provides details on functions, parameters, and data structures in the conferencing API.

Applicability

This document version (05-2505-002) is published for Dialogic® Host Media Processing (HMP) Software Release 3.1LIN (also referred to as Dialogic® HMP Software 3.1LIN).

This document may also be applicable to other software releases (including service updates) on Linux or Windows® operating systems. Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This publication is intended for the following audience:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)
- End Users

How to Use This Publication

This document assumes that you are familiar with the Linux or Windows® operating systems and the C programming language.

The information in this document is organized as follows:

- Chapter 1, "Product Description" introduces the key features of the conferencing software.
- Chapter 2, "Programming Models" provides a brief overview of supported programming models.
- Chapter 3, "Event Handling" provides information on functions used to handle events.
- Chapter 4, "Error Handling" provides information on handling errors in your application.
- Chapter 5, "Application Development Guidelines" provides programming guidelines for developing conferencing applications.
- Chapter 6, "Using Active Talker" provides details on using the active talker feature.
- Chapter 7, "Using Volume Control" provides details on using the volume control feature.
- Chapter 8, "Building Applications" discusses compiling and linking requirements such as includes files and library files.

Related Information

See the following for additional information:

- *http://www.dialogic.com/manuals/* (for Dialogic® product documentation)
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

Product Description 1

This chapter provides an overview of the Dialogic® Conferencing (CNF) API library. Topics include:

1.1 Overview

The Dialogic® Conferencing (CNF) API software supports development of conferencing applications on Dialogic® Host Media Processing (HMP) Software. The conference can take place over an IP network and/or over traditional public switched telephone network (PSTN) lines.

Dialogic® HMP Software performs media processing tasks on general-purpose servers based on Intel architecture without the need for specialized hardware. When installed on a system, Dialogic® HMP Software performs like a virtual Dialogic® DM3 board to the customer application, but media processing takes place on the host processor. In this document, the term "board" represents the virtual Dialogic® DM3 board.

Note: This Dialogic® Conferencing (CNF) API is distinct from and incompatible with the Dialogic® Conferencing (CNF) API that was previously released in Dialogic® System Release 6.0 on PCI for Windows®.

1.2 Key Features

Key features of the Dialogic® Conferencing (CNF) API software include the following:

Asynchronous programming model support
: This model enables multiple channels to be handled in a single process and supports higher density conferencing solutions.

Support for conferees from multiple sources
: Participants in a conference may come from a variety of sources, such as a voice device and an IP media device. The software allows for flexibility to grow and support additional sources.

Conference bridging
: Multiple conferences can be bridged together so that all parties (also called conferees) in two or more established conferences can communicate with one another.

Coach/pupil feature
: Two selected parties can establish a private communication link within the overall conference. The coach is a private member of the conference and is only heard by the pupil. However, the pupil cannot speak privately with the coach.

DTMF digit detection
: The application can determine whether a party has generated a DTMF digit.

Volume control
: A party can adjust the listening volume of the conference using pre-programmed DTMF digits.

DTMF tone clamping
: This feature mutes dual tone multi-frequency (DTMF) tones heard during a conference. Tone clamping applies to the transmitted audio going into the conference and does not affect DTMF function. It can be enabled on a board, conference, or party basis.

Automatic gain control (AGC)
: AGC is an algorithm for normalizing an input signal to a target level. The AGC algorithm discriminates between voiced and unvoiced signals within a conference.

Active talker
: The active talker feature sums the three most active talkers in a conference, so that the conversation doesn't get drowned out when too many people talk at once.

Conference monitoring
: Participants have listen-only access to a conference.

Echo cancellation
: This feature reduces echo from the incoming signal, improving the quality of a conference for all participants.

Tariff tone
: A party can receive a periodic tone for the duration of the conference call.

1.3 Understanding How Conferences are Formed

Developing a conferencing application requires the use of the Dialogic® Conferencing (CNF) API library as well as other Dialogic® API libraries, such as the Dialogic® Standard Runtime Library (SRL) and the Dialogic® Device Management API library. Other libraries include the IP media and voice libraries.

A conference consists of conferees (also known as parties). The maximum number of conferences and parties supported varies with the Dialogic® HMP Software license in use and, if applicable, the media load in use on the board.

A conference is identified by a unique conference device handle, which is registered with the Dialogic® Standard Runtime Library (SRL). A party is identified by a unique SRL party device handle. The virtual board device is the parent device for the conference device and party device; it has a unique SRL device handle. For more information on the types of conferencing devices, see Section 5.3, "Creating a Conference", on page 20.

The Dialogic® Conferencing (CNF) API is used to open a conference, and to add parties to a conference. However, these parties cannot participate in a conference until they are connected to a technology device handle through the **dev_Connect( )** Dialogic® Device Management API function. Technology device handles are obtained through the respective technology API library functions. For example, the dxxxB1C1 voice channel device handle is obtained from **dx_open( )**.

A conference may be formed from parties that are connected to any one of the following technology device handles:

- voice (dx) device handle
- IP media (ipm) device handle
- digital network interface (dti) device handle

Note: A device handle obtained from **gc_OpenEx( )** in the Dialogic® Global Call API library cannot be used by **dev_Connect( )** to connect a party to a conference. Rather, you can use the device handle returned by **gc_GetResourceH( )** to connect a party to a conference.

1.4 Relationship with Other Libraries

A conferencing application is developed using the Dialogic® Conferencing (CNF) API library as well as other Dialogic® API libraries, including the following:

- Dialogic® Standard Runtime Library (SRL)
- Dialogic® Device Management API Library
- Dialogic® Voice API Library
- Dialogic® IP Media Library API
- Dialogic® Global Call API Library
- Dialogic® Digital Network Interface API Library

1.4.1 Dialogic® Standard Runtime Library (SRL)

The Dialogic® Standard Runtime Library (SRL) provides a common interface for event handling and other functionality common to all devices.

The Dialogic® Conferencing (CNF) API uses three types of devices: virtual board device, conference device, and party device. TheDialogic® Conferencing (CNF) API registers the virtual board device with the Dialogic® Standard Runtime Library (SRL) when **cnf_Open( )** is called. In addition, the conference device and the party device are registered when **cnf_OpenConference( )** and **cnf_OpenParty( )**, respectively, are called. Conferencing events are posted to the SRL, which then delivers these events to the application. For more information about SRL functions, see the *Dialogic® Standard Runtime Library API Library Reference*.

1.4.2 Dialogic® Device Management API Library

The Dialogic® Device Management API library provides run-time control and management of configurable system devices. It includes functions to reserve resources and to manage the

connections between devices. It performs all necessary connection-related operations, including time slot management.

The device connection functions enable connection between conferencing devices and other devices on Dialogic® HMP Software, providing the ability for conferencing communication. Before a party can participate in a conference, it must be connected to a supported technology device (such as voice and IP media) using the **dev_Connect( )** function. Conference bridging is also accomplished through the Dialogic® Device Management API library. For more information about device management functions, see the *Dialogic® Device Management API Library Reference*.

1.4.3 Dialogic® Voice API Library

The Dialogic® Voice API provides a collection of functions supporting call processing such as dual tone multifrequency (DTMF) detection, tone signaling, playing and recording. You may add a party to a conference using a device handle obtained from **dx_open( )**. You must then connect the voice device to a conference using **dev_Connect( )**. For more information about voice functions, see the *Dialogic® Voice API Library Reference*.

1.4.4 Dialogic® IP Media Library API

The Dialogic® IP Media Library API provides a collection of functions for media control on IP devices. You may add a party to a conference using a device handle obtained from **ipm_Open( )**. You must then connect the IP media device to a conference using **dev_Connect( )**. For more information about Dialogic® IP Media Library API functions, see the *Dialogic® IP Media Library API Library Reference*.

1.4.5 Dialogic® Global Call API Library

The Dialogic® Global Call API provides a collection of functions supporting call control operations. You may add a party to a conference using a device handle obtained from **gc_GetResourceH( )**. You must then connect the device to a conference using **dev_Connect( )**. For more information about Dialogic® Global Call API functions, see the *Dialogic® Global Call API Library Reference*.

1.4.6 Dialogic® Digital Network Interface API Library

The Dialogic® Digital Network Interface API is used to manage digital network interface devices. You may add a party to a conference using a device handle obtained from **dt_open( )**. You must then connect the device to a conference using **dev_Connect( )**. For more information about Dialogic® Digital Network Interface API functions, see the *Dialogic® Digital Network Interface Software Reference*.

Programming Models 2

This chapter describes the programming models supported by the Dialogic® Conferencing API software. The following topics are covered:

2.1 Programming Models Overview

The Dialogic® Conferencing API software supports application development using asynchronous programming models. By usage, the asynchronous models are often said to use asynchronous **mode**. Asynchronous mode programming is introduced briefly in this chapter and described in more detail in the *Dialogic® Standard Runtime Library API Programming Guide*.

Note: The Dialogic® Conferencing API library is implemented as an asynchronous only library. If desired, you can implement synchronous functionality in the application itself.

2.2 Asynchronous Programming Model

Asynchronous mode programming is characterized by allowing other processing to take place while a function executes. In asynchronous mode programming, multiple channels are handled in a single process rather than in separate processes as required in synchronous mode programming.

An asynchronous mode function typically receives an event from the Dialogic® Standard Runtime Library (SRL) indicating completion (termination) of the function in order for the application to continue processing a call on a particular channel. A function called in the asynchronous mode returns control to the application after the request is passed to the device driver. A termination event is returned when the requested operation completes.

Caution: In general, when a function is called in asynchronous mode, and an associated termination event exists, the **cnf_Close( )** function should not be called until the termination event has been received.

For Linux environments, the asynchronous models provided for application development include:

Asynchronous (Polled)

In this model, the application polls for or waits for events using the **sr_waitevt( )** function. When an event is available, event information may be retrieved using SRL event handling functions such as **sr_getevttype( )**. Retrieved event information is valid until the **sr_waitevt( )** function is called again. Typically, the polled model is used for applications that do not need to use event handlers to process events.

Asynchronous with Event Handlers

This model may be run in non-signal mode only. Event handlers can be enabled or disabled for specific events on specific devices.

Event Handling 3

All conferencing events are retrieved using Dialogic® Standard Runtime Library (SRL) event retrieval mechanisms, including event handlers. The SRL is a device-independent library containing event management functions and Standard Attribute functions. This chapter lists SRL functions that are typically used by conferencing applications.

3.1 Dialogic® Standard Runtime Library Event Management Functions

SRL event management functions retrieve and handle device termination events for certain library functions. Applications typically use the following functions:

sr_enbhdlr( )
enables event handler

sr_dishdlr( )
disables event handler

sr_getevtdev( )
gets device handle

sr_getevttype( )
gets event type

sr_waitevt( )
waits for next event

sr_waitevtEx( )
waits for events on certain devices

Note: See the *Dialogic® Standard Runtime Library API Library Reference* for function details.

3.2 Dialogic® Standard Runtime Library Standard Attribute Functions

SRL Standard Attribute functions return general device information, such as the device name or the last error that occurred on the device. Applications typically use the following functions:

ATDV_ERRMSGP( )
pointer to string describing the error that occurred during the last function call on the specified device

ATDV_LASTERR( )
error that occurred during the last function call on a specified device. See the function description for possible errors for the function.

ATDV_NAMEP( )
pointer to device name

ATDV_SUBDEVS( )
number of subdevices

Note: See the *Dialogic® Standard Runtime Library API Library Reference* for function details.

Error Handling 4

This chapter describes error handling for the Dialogic® Conferencing (CNF) API software.

All Dialogic® Conferencing (CNF) API functions return a value that indicates the success or failure of the function call. Success is indicated by a return value of CNF_SUCCESS. Failure is indicated by a value of CNF_ERROR.

If a function fails, call the Dialogic® Standard Runtime Library API functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** for the reason for failure. These functions are described in the *Dialogic® Standard Runtime Library API Library Reference*.

If an error occurs during execution of an asynchronous function, the CNFEV_ERROR event is sent to the application. No change of state is triggered by this event. Upon receiving the CNFEV_ERROR event, the application can retrieve the reason for the failure using the Dialogic® Standard Runtime Library API functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )**.

Application Development Guidelines 5

This chapter contains guidelines for developing Dialogic® Conferencing (CNF) API applications. The following topics are covered:

5.1 Using Symbolic Defines

The numerical values of defines may not remain the same as new versions of the software are released. It is recommended that you do not use a numerical value in your application when an equivalent symbolic define is available. Symbolic defines are found in the header files; for example, *cnflib.h*, *cnfevts.h, cnferrs.h,* and *srllib.h*.

5.2 Using Conferencing Devices

The types of devices used in the Dialogic® Conferencing (CNF) API library and their naming convention are as follows:

- virtual board device, called cnfB*x*, where *x* is the logical board number
- conference device, called cnfB*x*C*y*, where *x* is the logical board number and *y* is the conference device channel
- party device, called ptyB*x*P*z*, where *x* is the logical board number and *z* is the party device channel

All devices are identified by a unique SRL handle. All subsequent references to the opened device must be made using the handle, until the device is closed.

The virtual board device is the parent device for both the conference device and the party device. You must open a virtual board device before opening a conference device or party device. After a board device is opened, you can open and initialize all conference devices at once, and/or all party

devices at once. A conference device and a party device are independent; that is, you can open a party device without first opening a conference device.

5.3 Creating a Conference

The following steps describe how to create a conference. See the Glossary for information on the terms used here. See the *Dialogic® Conferencing API Library Reference* for details on conferencing functions and data structures.

Note: These steps provide general guidelines. They do not cover all tasks required to write a conferencing application.

1. Use the asynchronous programming model, and enable a Dialogic® Standard Runtime Library (SRL) event handler for the various devices used by the conferencing software (virtual board, conference, and party) via **sr_enbhdlr( )**.
2. Open the virtual board device handle using **cnf_Open( )**. The device naming convention for the virtual board is **cnfBx**, where **x** is the board number starting at 1. You must have a virtual board device before you can open a conference device or a party device.
3. Get a count of the resources on this board using **cnf_GetDeviceCount( )**. This count is a snapshot in time. The CNF_DEVICE_COUNT_INFO data structure contains information about the number of devices on this board, such as the maximum number of conferences and parties, as well as the number of free conferences and free parties. The maximum number of conferences and parties supported varies with the Dialogic® Host Media Processing (HMP) Software license in use and, if applicable, the media load in use on the board. Having a count of the resources enables you to properly manage these resources.
4. If desired, specify attributes for the board using **cnf_SetAttributes( )**. Attributes are contained in the CNF_ATTR data structure. Use **cnf_GetAttributes( )** to return the current attributes for the board.
5. If desired, enable notification events for the board using **cnf_EnableEvents( )**. Events are contained in the CNF_EVENT_INFO data structure. For example, the application can be notified dynamically whenever a conference is opened or a party is added.
6. At this point, you can choose to open and set up all conferences; or you can choose to open one conference at a time as needed. Similarly, you can also choose to open and set up all parties, or open one party at a time as needed. The steps that follow show how to open one conference, then add a party to this opened conference. Repeat the steps as appropriate for your use case.
7. Using **cnf_OpenConference( )**, create a new conference to which parties will be added. This function takes the virtual board device handle returned by **cnf_Open( )** as an argument. It returns a unique SRL device handle for the conference. The conference created consumes a conference resource.
8. If desired, specify attributes for the conference using **cnf_SetAttributes( )**. Attributes are contained in the CNF_ATTR data structure. Use **cnf_GetAttributes( )** to return the current attributes for the conference.
9. If desired, enable notification events for the conference using **cnf_EnableEvents( )**. Events are contained in the CNF_EVENT_INFO data structure.
10. Open a party device handle using **cnf_OpenParty( )**. This function returns a unique SRL device handle for the party.

11. If desired, you can specify attributes for a party using **cnf_SetAttributes( )**. Attributes are contained in the CNF_ATTR data structure. Use **cnf_GetAttributes( )** to return the current attributes for the party.
12. Before a party can participate in a conference, you must connect this party to a supported technology device using **dev_Connect( )**. Examples of supported technology devices include a voice device (dxxxB1C1) and an IP device (ipmB1C1). See Section 1.3, "Understanding How Conferences are Formed", on page 10 for details on supported technology devices. See the *Dialogic® Device Management API Library Reference* for details on device management functions.

    Note: Depending on your use case, you can choose to issue **dev_Connect( )** either before or after performing the **cnf_AddParty( )** operation in Step 13. If you issue **dev_Connect( )** after adding a party, you must wait for this function to successfully complete before streaming can take place.
13. Using **cnf_AddParty( )**, add a party to the conference created in step 7. This function takes the party device handle returned by **cnf_OpenParty( )** as an argument. The party created consumes a party resource.
14. Add more parties to the conference as needed. There is a limit to the number of parties that can be added to a conference (the count of resources was obtained in step 3). However, if the limit is reached, you can add parties using the conference bridging feature. For more information on bridging, see Section 5.4, "Conference Bridging", on page 21.
15. Terminate your application in an orderly fashion. For example, disable events, close all devices, and so on. For more information, see Section 5.5, "Terminating an Application", on page 22.

5.4 Conference Bridging

If a conference expands beyond the number of parties permitted by the Dialogic® HMP Software license in use and, if applicable, the media load in use on the board, you can create a second conference to support additional conferees. The two conferences are connected via a conference bridge. Conference bridging allows all parties in two or more conferences to speak with and/or listen to one another.

The following guidelines for creating a conference bridge assume that you have already created two conferences and added the desired number of parties for each conference using the instructions in Section 5.3, "Creating a Conference", on page 20.

- Dedicate a party (party1) in conference A to serve as the bridge to conference B. Likewise, dedicate a party (party2) in conference B to serve as the bridge to conference A.
- Connect party1 in conference A to party2 in conference B using **dev_Connect( )**, a function in the Dialogic® Device Management API library. See the *Dialogic® Device Management API Library Reference* for details on device management functions.

The following rules apply to conference bridging:

- Each bridge that is created consumes two licensed party resources, one from each of the conferences involved in the bridge.

- Even though two (or more) conferences can be bridged together, the attributes and settings of each conference remain unchanged. The application is responsible for managing each conference and conference related events separately.
- The coach/pupil feature does not span conference bridges. Coach and pupil must be in the same conference.

5.5 Terminating an Application

Party resources and conference resources are not released when an application terminates. The conferencing software is designed in this way to allow conferences to stay active when a process exits. Therefore, you are responsible for terminating the application properly. Similarly, if an error condition abnormally terminates the application, individual conferences will not be closed nor will individual channels be closed. In this case, design the application to recover and manage the existing conferences or to shut down devices in an orderly fashion.

When your process completes, devices should be shut down in an orderly fashion. Tasks that are performed to terminate an application generally include:

- disabling events by calling **cnf_DisableEvents( )**
- closing all devices using the appropriate function such as **cnf_CloseParty( )**, **cnf_CloseConference( )**, **cnf_Close( )**, **dx_close( )**, and so on
- breaking the connection between the party device and other supported device using **dev_Disconnect( )**

Note: Standard Runtime Library event management functions (such as **sr_dishdlr( )**, which disables an event handler) must be called before closing the device that is sending the handler event notifications. See Chapter 3, "Event Handling" for more information about handling events.

5.6 Data Structure Considerations

Take note of the following consideration when working with data structures:

- Each data structure in the conferencing library has a version number field. This version number is used to ensure that an application is binary compatible with future changes to this data structure. This field is currently reserved for future use. Use the version number as specified in the header file, *cnflib.h*, and as documented in the *Dialogic® Conferencing API Library Reference*.

5.7 Multiprocessing Considerations

Having multiple processes acting on the same board is undesirable. It is recommended to use a single process per board, or a single process for all boards, rather than more than one process acting on the same board. Consider the scenario where there are multiple boards in the system and each board is being controlled by a different process.

The following considerations apply when multiple processes control the same board:

- You must provide your own synchronization to manage resources in each process.
- If process A creates a conference and process B wants to use that conference, process A must pass the name of the conference to process B.
- If process A deletes a conference and process B has a handle to that conference, then process B can no longer use that conference. Process A must notify process B of its action.

5.8 Multithreading Considerations

The following considerations apply to multithreading:

- The conferencing library supports multithreading. You can manage multiple conferences or multiple boards within the same thread; however, it is not recommended that you manage the same conference or the same board across multiple threads.
- The resource counts returned by **cnf_GetDeviceCount( )** are a snapshot in time. If another thread is adding/deleting a party or creating/deleting a conference, the counts will change and the thread will no longer have the most current count. There is a gap between the time you issue this function and when you actually use the resources. Be sure that threads use synchronization when making decisions based on the counts returned by **cnf_GetDeviceCount( )**.
- While the API functions allow for concurrent use of party, conference and board handles, you must be aware of "logical" concurrency issues, such as maintaining the count of resources.

  The **cnf_GetDeviceCount( )** function returns a snapshot of available parties and maximum parties that can be added to a conference. Because it is a snapshot of the state of the firmware at any given time, the values returned are only valid until other parties and conferences are added or removed.

  In a multithreaded application, you should maintain local counts that are obtained when the application initializes (through **cnf_GetDeviceCount( )**) and protect those counts with mutexes as needed; for example, if two or more threads in the application need to make decisions based on the number of parties and conferences available at any given time. By doing so, race conditions can be avoided; for example, if a thread thinks one more party resource is available while another thread consumes it.

Using Active Talker 6

This chapter provides information about the active talker feature.

An active talker refers to a party in a conference who is providing “non-silence” energy. Active talkers are determined by the loudness or strength of their “non-silence” energy. The active talker feature sums the three most active talkers in a conference, so that the conversation doesn’t get drowned out when too many people talk at once. The active talker feature also provides data on active talkers through the **cnf_GetActiveTalkerList( )** function.

The active talker feature is enabled on a board basis. To turn on the active talker feature, use **cnf_SetAttributes( )** with the ECNF_BRD_ATTR_ACTIVE_TALKER enumeration enabled. To retrieve a list of active talkers, use **cnf_GetActiveTalkerList( )**.

Note: The active talker feature does not span conference bridges; that is, there is no active talker summing across conference bridges and active talkers are reported separately for each conference.

The **cnf_GetActiveTalkerList( )** function provides a snapshot of the active talkers at a given moment. By default, the snapshot is updated every second. To change this value and specify how frequently the active talker status is updated, use the **cnf_SetAttributes( )** function with the ECNF_BRD_ATTR_NOTIFY_INTERVAL enumeration and specify a value in 10 msec units. If a low value is used, it can affect system performance due to the more frequent updating of the status (which results in a high quantity of internal notification messages). If a high value is used, it will result in less frequent updating on active talkers, but the non-silence energy by a conferee may not be reported if it occurs between notification updates. For example, if the notification interval is set to 2 seconds and a conferee only says “yes” or “no” quickly in between notifications, that vocalization by the conferee will not be reported.

Using Volume Control 7

This chapter provides information about controlling the volume level in a conference.

A party in a conference may wish to change the volume level of the received signal. This is accomplished using the volume control feature.

The **cnf_SetDTMFControl( )** function allows the application to define the DTMF digits that cause the volume level to be adjusted up, down, or back to the default. This function points to the CNF_DTMF_CONTROL_INFO structure which specifies whether volume control is enabled or not and contains details on the digits used for volume control. Volume control is enabled on a board basis.

The **cnf_GetDTMFControl( )** function returns information on the DTMF digits used to control the volume.

The default volume or origin is 0 dB. Volume is incremented or decremented by 2 dB at a time. The maximum value for the volume is 18 dB and the minimum value is -18 dB.

Building Applications 8

This chapter provides information on building applications using the Dialogic® Conferencing (CNF) API library. The following topics are discussed:

8.1 Compiling and Linking

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

8.1.1 Include Files

Function prototypes and symbolic defines are determined in include files, also known as header files. Applications that use Dialogic® Conferencing (CNF) API library functions must contain statements for include files in this form, where <filename> represents the include file name:

```
#include <filename.h>
```

The following header files must be included in the application code **in the order shown** prior to calling the Dialogic® Conferencing (CNF) API library functions:

srllib.h
Contains function prototypes and equates for the Dialogic® Standard Runtime Library.

Note: *srllib.h* must be included in code before all other Dialogic® header files.

cnflib.h
The primary header file for the Dialogic® Conferencing (CNF) API library. Contains function prototypes and symbolic defines.

cnferrs.h
Contains equates for conferencing error codes.

cnfevts.h
Contains equates for conferencing event codes.

devmgmt.h
Contains function prototypes and symbolic defines for the Dialogic® Device Management API library.

If you use other library functions such as voice or IP media, you will have to include the header files for that library:

dxxxlib.h
Contains function prototypes and symbolic defines for the Dialogic® Voice API library.

dtilib.h
Contains function prototypes and symbolic defines for the Dialogic® Digital Network Interface API library.

gclib.h
The primary header file for the Dialogic® Global Call API library; contains function prototypes and symbolic defines for this library.

ipmerror.h
Contains variables for Dialogic® IP Media Library API error codes.

ipmlib.h
Contains function prototypes and symbolic defines for the Dialogic® IP Media Library API.

8.1.2 Required Libraries

Windows®

In Windows®, you must link the following library files when compiling your conferencing application:

libsrlmt.lib
Dialogic® Standard Runtime Library API file. Required in all applications.

libdxxmt.lib
Dialogic® Device Management API library file. Required only if the application uses Dialogic® Voice API library functions directly; for example, **dx_open( )**.

libdtimt.lib
Dialogic® Digital Network Interface API library file. Required only if the application uses Dialogic® Digital Network Interface API library functions directly; for example, **dt_open( )**.

libgc.lib
the primary Dialogic® Global Call APIlibrary file. Required only if the application uses Dialogic® Global Call API library functions directly; for example, **gc_GetResourceH( )**.

libipm.lib
the primary Dialogic® IP Media Library API file. Required only if the application uses Dialogic® IP Media Library API functions directly; for example, **ipm_Open( )**.

libdevmgmt.lib
Dialogic® Device Management API library file. Required in a conferencing application.

libcnf.lib
Dialogic® Conferencing (CNF) API library file. Required in a conferencing application.

Linux

In Linux, you must link the following library files in the order shown when compiling your conferencing application:

libsrl.so
Dialogic® Standard Runtime Library API file. Required in all applications. Specify `-lsrl` in makefile.

libdxxx.so
the primary Dialogic® Voice API library file. Required only if the application uses Dialogic® Voice API library functions directly; for example, **dx_open( )**. Specify `-ldxxx` in makefile.

libdti.so
Dialogic® Digital Network Interface API library file. Required only if the application uses Dialogic® Digital Network Interface API library functions directly; for example, **dt_open( )**. Specify `-ldti` in makefile.

libgc.so
the primary Dialogic® Global Call APIlibrary file. Required only if the application uses Dialogic® Global Call API library functions directly; for example, **gc_GetResourceH( )**. Specify `-lgc` in makefile.

libipm.so
the primary Dialogic® IP Media Library API file. Required only if the application uses Dialogic® IP Media Library API functions directly; for example, **ipm_Open( )**. Specify `-lipm` in makefile.

libdevmgmt.so
Dialogic® Device Management API library file. Required in a conferencing application. Specify `-ldevmgmt` in makefile.

libcnf.so
Dialogic® Conferencing (CNF) API library file. Required in a conferencing application. Specify `-lcnf` in makefile.

By default, the library files are located in the directory given by the INTEL_DIALOGIC_LIB environment variable.

8.2 Variables for Compiling and Linking

The following variables provide a standardized way of referencing the directories that contain header files and shared objects:

INTEL_DIALOGIC_INC
Variable that points to the directory where header files are stored.

INTEL_DIALOGIC_LIB
Variable that points to the directory where shared library files are stored.

These variables are automatically set at login and should be used in compiling and linking commands. The following is an example of a compiling and linking command that uses these variables:

```
cc -I${INTEL_DIALOGIC_INC} -o myapp myapp.c -L${INTEL_DIALOGIC_LIB} -lcnf -srl
```

Note: It is strongly recommended that you use these variables when compiling and linking applications. The name of the variables will remain constant, but the values may change in future releases.

Glossary

active talker: A participant in a conference who is providing “non-silence” energy.

automatic gain control (AGC): An electronic circuit used to maintain the audio signal volume at a constant level. AGC maintains nearly constant gain during voice signals, thereby avoiding distortion, and optimizes the perceptual quality of voice signals by using a new method to process silence intervals (background noise).

asynchronous function: A function that allows program execution to continue without waiting for a task to complete. To implement an asynchronous function, an application-defined event handler must be enabled to trap and process the completed event. Contrast with synchronous function.

bit mask: A pattern which selects or ignores specific bits in a bit-mapped control or status field.

bitmap: An entity of data (byte or word) in which individual bits contain independent control or status information.

board device: A board-level object that maps to a virtual board.

buffer: A block of memory or temporary storage device that holds data until it can be processed. It is used to compensate for the difference in the rate of the flow of information (or time occurrence of events) when transmitting data from one device to another.

bus: An electronic path that allows communication between multiple points or devices in a system.

busy device: A device that has one of the following characteristics: is stopped, being configured, has a multitasking or non-multitasking function active on it, or I/O function active on it.

channel device: A channel-level object that can be manipulated by a physical library, such as an individual telephone line connection. A channel is also a subdevice of a board.

CO (Central Office): A local phone network exchange, the telephone company facility where subscriber lines are linked, through switches, to other subscriber lines (including local and long distance lines). The term “Central Office” is used in North America. The rest of the world calls it “PTT”, for Post, Telephone, and Telegraph.

coach: A participant in a conference that can be heard by pupils only. A mentoring relationship exists between a coach and a pupil.

conferee: Participant in a conference call. Synonym of party.

conference: Ability for three or more participants in a call to communicate with one another in the same call.

conferencing: Ability to perform a conference.

conference bridging: Ability for all participants in two or more established conferences to speak to and/or listen to one another.

configuration file: An unformatted ASCII file that stores device initialization information for an application.

configuration manager: A utility with a graphical user interface (GUI) that enables you to add new boards to your system, start and stop system service, and work with board configuration data. Also known as DCM.

CT Bus: Computer Telephony bus. A time division multiplexing communications bus that provides 4096 time slots for transmission of digital information between CT Bus products. See TDM bus.

data structure: Programming term for a data element consisting of fields, where each field may have a different type definition and length. A group of data structure elements usually share a common purpose or functionality.

device: A computer peripheral or component controlled through a software device driver. A Dialog® voice and/or network interface expansion board is considered a physical board containing one or more logical board devices, and each channel or time slot on the board is a device.

device channel: A voice data path that processes one incoming or outgoing call at a time (equivalent to the terminal equipment terminating a phone line).

device driver: Software that acts as an interface between an application and hardware devices.

device handle: Numerical reference to a device, obtained when a device is opened using **xx_open( )**, where *xx* is the prefix defining the device to be opened. The device handle is used for all operations on that device.

device name: Literal reference to a device, used to gain access to the device via an **xx_open( )** function, where *xx* is the prefix defining the device to be opened.

DM3: Refers to Dialogic® mediastream processing architecture, which is open, layered, and flexible, encompassing hardware as well as software components. A whole set of products from Dialogic are built on DM3 architecture.

driver: A software module which provides a defined interface between a program and the firmware interface.

DTMF (Dual-Tone Multifrequency): Push-button or touch-tone dialing based on transmitting a high- and a low-frequency tone to identify each digit on a telephone keypad.

E1: A CEPT digital telephony format devised by the CCITT, used in Europe and other countries around the world. A digital transmission channel that carries data at the rate of 2.048 Mbps (DS-1 level). CEPT stands for the Conference of European Postal and Telecommunication Administrations. Contrast with T1.

extended attribute functions: A class of functions that take one input parameter and return device-specific information. For instance, a voice device's extended attribute function returns information specific to the voice devices. Extended attribute function names are case-sensitive and must be in capital letters. See also standard runtime library (SRL).

firmware: A set of program instructions that reside on an expansion board.

idle device: A device that has no functions active on it.

party: A participant in a conference. Synonym of conferee.

pupil: A participant in a conference that has a mentoring relationship with a coach.

resource: Functionality (for example, conferencing) that can be assigned to a call. Resources are *shared* when functionality is selectively assigned to a call and may be shared among multiple calls. Resources are *dedicated* when functionality is fixed to the one call.

RFU: Reserved for future use.

route: Assign a resource to a time slot.

SRL: See **Standard Runtime Library**.

standard attribute functions: Class of functions that take one input parameter (a valid device handle) and return generic information about the device. For instance, standard attribute functions return IRQ and error information for all device types. Standard attribute function names are case-sensitive and must be in capital letters. Standard attribute functions for all Dialogic® devices are contained in the SRL. See standard runtime library (SRL).

standard runtime library (SRL): A Dialogic® software resource containing event management and standard attribute functions and data structures used by all Dialogic® devices, but which return data unique to the device.

synchronous function: Blocks program execution until a value is returned by the device. Also called a blocking function. Contrast with asynchronous function.

T1: A digital line transmitting at 1.544 Mbps over 2 pairs of twisted wires. Designed to handle a minimum of 24 voice conversations or channels, each conversation digitized at 64 Kbps. T1 is a digital transmission standard in North America. Contrast with E1.

TDM (Time Division Multiplexing): A technique for transmitting multiple voice, data, or video signals simultaneously over the same transmission medium. TDM is a digital technique that interleaves groups of bits from each signal, one after another. Each group is assigned its own "time slot" and can be identified and extracted at the receiving end. See also time slot.

TDM bus: Time division multiplexing bus. A resource sharing bus such as the SCbus or CT Bus that allows information to be transmitted and received among resources over multiple data lines.

termination condition: An event or condition which, when present, causes a process to stop.

termination event: An event that is generated when an asynchronous function terminates. See also asynchronous function.

thread (Windows®): The executable instructions stored in the address space of a process that the operating system actually executes. All processes have at least one thread, but no thread belongs to more than one process. A multithreaded process has more than one thread that are executed seemingly simultaneously. When the last thread finishes its task, then the process terminates. The main thread is also referred to as a primary thread; both main and primary thread refer to the first thread started in a process. A thread of execution is just a synonym for thread.

tone clamping: (DTMF tone clamping) Mutes DTMF tones heard in a conference. If a confereee's phone generates a tone, the DTMF signal will not interfere with the conference. Applies to transmitted audio into the conference and does not affect DTMF function.

time division multiplexing (TDM): See TDM (Time Division Multiplexing).

time slot: The smallest, switchable data unit on a TDM bus. A time slot consists of 8 consecutive bits of data. One time slot is equivalent to a data path with a bandwidth of 64 kbps. In a digital telephony environment, a normally continuous and individual communication (for example, someone speaking on a telephone) is (1) digitized, (2) broken up into pieces consisting of a fixed number of bits, (3) combined with pieces of other individual communications in a regularly repeating, timed sequence (multiplexed), and (4) transmitted serially over a single telephone line. The process happens at such a fast rate that, once the pieces are sorted out and put back together again at the receiving end, the speech is normal and continuous. Each individual, pieced-together communication is called a time slot.

Index

L

M

N

P

R

S

T

V