Conferencing API for Linux and Windows

Programming Guide

November 2003

05-2067-001

Publication Date: November 2003

Document Number: 05-2067-001

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

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

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

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

Contents

Figures

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2067-001	November 2003	Initial version of document.

About This Publication

The following topics provide information about this *Conferencing API for Linux and Windows Programming Guide*:

Purpose

This publication provides guidelines for developing applications for Intel® telecom products using the conferencing (CNF) API. It is a companion document to the *Conferencing API Library Reference* which provides a reference to all functions, parameters, and data structures in the conferencing API. The conferencing API is supported in Linux* or Windows* operating systems on Intel® telecom products that implement the DM3 architecture.

Note: This conferencing API is intended to replace the existing audio conferencing (DCB) API. Although the DCB API continues to be supported, it is recommended that all new conferencing applications be developed using the new conferencing (CNF) API as no further development is planned on the DCB API.

Intended Audience

This publication is intended for:

- 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 system and the C programming language.

The information in this document is organized as follows:

- Chapter 1, "Product Description" describes the conferencing library, key features, and its relationship with other libraries such as the Standard Runtime Library.
- Chapter 2, "Programming Models" provides a brief overview of supported programming models.
- Chapter 3, "Device Handling" discusses topics related to devices such as device naming concepts as well as conferencing device names and handles.
- Chapter 4, "Event Handling" provides information on functions used to handle events.
- Chapter 5, "Error Handling" provides information on handling errors in your application.
- Chapter 6, "Application Development Guidelines" discusses programming guidelines and techniques for developing an application using the conferencing library.
- Chapter 7, "Active Talker" describes how to use the active talker feature.
- Chapter 8, "Conference Bridging" provides details on how to create conference bridges.
- Chapter 9, "Volume Control" describes how to use the volume control feature.

A glossary and an index are provided for your reference.

Related Information

See the following for more information:

- For details on all functions and data structures in the conferencing (CNF) library, see the *Conferencing API Library Reference*.
- For information about voice library features and guidelines for building applications using the voice library, see the *Voice API Programming Guide*.
- For details on all functions and data structures in the voice library, see the *Voice API Library Reference*.
- For information about Standard Runtime Library features and guidelines for building all applications, see the *Standard Runtime Library API Programming Guide*.
- For details on all functions and data structures in the Standard Runtime Library, see the *Standard Runtime Library API Library Reference*.
- For information on the system release, system requirements, software and hardware features, supported hardware, and release documentation, see the Release Guide for the system release you are using.
- For details on compatibility issues, restrictions and limitations, known problems, and late-breaking updates or corrections to the release documentation, see the Release Update.

  Be sure to check the Release Update for the system release you are using for any updates or corrections to this publication. Release Updates are available on the Telecom Support Resources website at *http://resource.intel.com/telecom/support/releases/index.html*.
- For details on configuration files (including CONFIG/FCD/PCD files) and instructions for configuring products, see the Configuration Guide for your product or product family.

- For technical support, see *http://developer.intel.com/design/telecom/support/*. This website contains developer support information, downloads, release documentation, technical notes, application notes, a user discussion forum, and more.
- For product and services information, see *http://www.intel.com/network/csp/*.

Product Description 1

This chapter describes the key features and capability of the conferencing software.

1.1 Conferencing Overview and Key Features

The conferencing (CNF) library provides a high-level interface to Intel® telecom boards that are based on DM3 architecture and is a building block for developing conferencing applications.

The conferencing software consists of a C language interface, device drivers and firmware.

To use the conferencing software on a board, you must configure the board with a media load that supports conferencing resources. For complete information about media loads, see the Configuration Guide for boards based on DM3 architecture.

Key features of the conferencing software include:

- conference bridging in which conferences can be linked together so that all parties in two or more established conferences can speak to and listen to one another, enabling large conferences to be built
- conference monitoring in which participants have listen-only access to a conference
- active talker status used to indicate which participants in a given conference are currently talking
- coach and pupil role assignments which can be used for training or for private, one-way messages such as setting up a sub-conference
- tone clamping which mutes dual tone multi-frequency (DTMF) tones generated by a given party
- automatic gain control (AGC) to equalize the volume levels of different parties
- volume control to enable an individual party to adjust the listening volume of the conference
- DTMF digit detection used to determine when a party has generated a DTMF digit
- echo cancellation which reduces echo from the incoming signal

1.2 Basic Conferencing Concepts

Before using the conferencing software, you should understand the following basic conferencing concepts:

conference
: Ability for two or more participants in a telephone call to speak to and listen to one another in the same call.

party
: Also called conferee. A participant in a conference.

conference resource
: An instance of a conference provided by the conferencing firmware. Each instance consumes one conference resource in a party group.

party resource
: An instance of a party provided by the conferencing firmware. Each instance consumes one party resource in a party group.

party group
: A collection of resources on a board. A party group consists of a number of party resources and conference resources. The number of party groups on a board is determined when you start the board in the configuration manager (DCM). A party group is a way to manage resources on a board.

third party
: Also called an external party. A party supported by a board that is not manufactured by Intel.

coach
: A participant in a conference who is only heard by a party designated as “pupil.” A mentoring relationship exists between a coach and a pupil. This set up allows for private conversations between the coach and the pupil. The coaching feature can be used for training or for private, one-way messages such as setting up a sub-conference.

pupil
: A participant in a conference who is mentored by a coach.

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

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

1.3 Relationship with Other Libraries

The conferencing library interacts with other Intel® Dialogic® libraries such as:

- Standard Runtime Library (SRL) (required)
- voice API library
- digital network interface API library

- IP media library
- Global Call API library

The conferencing API registers the conferencing device with the Standard Runtime Library (SRL) when **cnf_Open( )** is called. Conferencing events are posted to the SRL, which then delivers these events to the application. You must either call the **sr_enbhdlr( )** function to install an event handler or call the **sr_waitevt( )** function to retrieve and process the events posted by the conferencing API. Use the SRL functions to simplify application development by writing common event handlers to be used by all devices. For more information about SRL functions, see the *Standard Runtime Library API Library Reference*.

The voice API provides a collection of functions supporting call processing such as dual tone multifrequency (DTMF) detection, tone signaling, playing and recording. After creating a conference using the conferencing API, you may add a party to the conference from a voice resource. In this case, you use **dx_open( )** to open the voice device and obtain the voice device handle. You might also use the dx_ routing functions such as **dx_getxmitslot( )** to manage routing over the TDM bus. For more information about voice functions, see the *Voice API Library Reference*.

The 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( )**. For more information about digital network interface functions, see the *Digital Network Interface Software Reference*.

The IP media library (IPML) 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( )**. If you have a Global Call application, you can add IP resources to a conference using the handle obtained from **gc_GetResourceH( )**. For more information about IP media functions, see the *IP Media Library API Library Reference*.

The 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( )**. For more information about Global Call functions, see the *Global Call API Library Reference*.

Programming Models 2

This chapter briefly discusses the Standard Runtime Library and supported programming models:

2.1 Standard Runtime Library

The Standard Runtime Library (SRL) provides a set of common system functions that are device independent and are applicable to all Intel® Dialogic® devices. The SRL consists of a data structure, event management functions, device management functions (called standard attribute functions), and device mapper functions. You can use the SRL to simplify application development, such as by writing common event handlers to be used by all devices.

For more information on the Standard Runtime Library, see the *Standard Runtime Library API Library Reference* and *Standard Runtime Library API Programming Guide*.

2.2 Asynchronous Programming Model

Asynchronous programming enables a single program to control multiple conferencing channels within a single process. This allows the development of complex applications where multiple tasks must be coordinated simultaneously.

The asynchronous programming model uses functions that do not block thread execution; that is, the function continues processing under the hood. A Standard Runtime Library (SRL) event later indicates function completion.

Generally, if you are building applications that use any significant density, you should use the asynchronous programming model to develop field solutions.

For complete information on asynchronous programming models, see the *Standard Runtime Library API Programming Guide*.

2.3 Synchronous Programming Model

The synchronous programming model uses functions that block application execution until the function completes. This model requires that each channel be controlled from a separate process. This allows you to assign distinct applications to different channels dynamically in real time.

Synchronous programming models allow you to scale an application by simply instantiating more threads or processes (one per channel). This programming model may be easy to encode and manage but it relies on the system to manage scalability. Applying the synchronous programming model can consume large amounts of system overhead, which reduces the achievable densities and negatively impacts timely servicing of both hardware and software interrupts. Using this model, a developer can only solve system performance issues by adding memory or increasing CPU speed or both. The synchronous programming models may be useful for testing or very low-density solutions.

For complete information on synchronous programming models, see the *Standard Runtime Library API Programming Guide*.

Device Handling 3

This chapter describes the concept of a device and how devices are named and used.

3.1 Device Concepts

The following concepts are key to understanding devices and device handling:

device
: A computer component controlled through a software device driver. A resource board, such as a voice resource, conferencing resource, and network interface board, contains one or more logical board devices. Each channel or time slot on the board is considered a device.

device name
: A 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. For example, “dx” is the prefix for voice device, “cnf” for conferencing device, and so on.

device handle
: A 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. The conferencing library distinguishes between two types of device handles: a physical board device handle and a conference device handle.

physical and virtual boards
: The APIs functions distinguish between physical boards and virtual boards. The device driver views a single physical voice board with more than four channels as multiple emulated D/4x boards. These emulated boards are called virtual boards. For example, a DM/V480A-2T1 board with 48 channels of voice processing and two T1 trunk lines contains 12 virtual voice boards and two virtual network interface boards.

3.2 Conferencing Device Names and Handles

The Intel® Dialogic® system software assigns a device name to each device or each component on a board.

The conferencing library recognizes two types of device handles: a physical board device handle and a conference device handle.

A **physical board device handle** is a numerical reference to a physical board. A physical board device handle is a concept introduced in System Release 6.0. Previously there was no way to identify a physical board but only the virtual boards that make up the physical board. Having a physical board device handle enables API functions to act on all devices on the physical board. The

physical board device handle in a conferencing application is obtained from the **cnf_Open( )** function call. This name is used as input to several conferencing functions such as **cnf_AttachConference( )**, **cnf_CreateConference( )**, and **cnf_GetResourceCount( )**.

A physical board device handle is named **brdBn**, where **n** is the board number assigned in sequential order starting with 1.

A **conference device handle** is a numerical reference to a conferencing device on a physical board. The conference device handle is obtained from the **cnf_CreateConference( )** or **cnf_AttachConference( )** function calls. This conference device handle is then used when adding parties to a conference, removing parties from a conference, and so on.

A conferencing device handle is named **cnfBnCy**, where **n** is the board number assigned in sequential order starting with 1 and **y** corresponds to one of the conference instances. The conference device handle consists of a hexadecimal number.

For complete information on device handling, see the *Standard Runtime Library API Programming Guide*.

Event Handling 4

This chapter provides a brief introduction to events and event handling.

An event indicates that a specific activity has occurred on a channel. The driver reports channel activity to the application program in the form of events, which allows the program to identify and respond to a specific occurrence on a channel. Events provide feedback on the progress and completion of functions and indicate the occurrence of other channel activities. Conferencing library events are defined in the *cnflib.h* header file.

For a list of events that may be returned by the conferencing software, see the *Conferencing API Library Reference*.

If your application is running in asynchronous mode, you may need to use a Standard Runtime Library (SRL) function to collect events being sent from the firmware, depending on the programming model in use. For more information, see the *Standard Runtime Library API Programming Guide*.

Error Handling 5

This chapter briefly describes how to handle errors that occur when running conferencing applications.

Conferencing library functions return a value to indicate success or failure of the function:

- To indicate success, the library returns CNF_SUCCESS.
- To indicate failure, the library returns CNF_FAILURE.

These constants are defined in *cnflib.h*. Error codes that may be returned by a function are described in the *Conferencing API Library Reference*.

If a library function fails, call the standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return a string describing the error. These functions are described in the *Standard Runtime Library API Library Reference*.

Application Development Guidelines 6

This chapter provides programming guidelines and techniques for developing an application using the conferencing library. The following topics are discussed:

6.1 Using Symbolic Defines

The system software does not guarantee the numerical values of defines will remain the same as new versions of the software package are released. In general, do not use a numerical value in your application when an equivalent symbolic define is available. Symbolic defines are found in the header files, such as *cnflib.h* and *srllib.h*.

6.2 Managing Resources

This section provides information on resource management in a conferencing application:

- Getting a Resource Count and Creating a Conference
- Sample Scenario for Managing Resources
- Resource Management Rules

6.2.1 Getting a Resource Count and Creating a Conference

Follow the steps in this section to manage resources on a board and create a conference. See Section 1.2, “Basic Conferencing Concepts”, on page 14 for information on the terms used here.

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

1. Assuming that you are using the asynchronous programming model (the recommended model), enable a Standard Runtime Library (SRL) event handler for the conferencing device via **sr_enbhdlr( )**.
2. Open the physical board device handle using **cnf_Open( )**. The device naming convention for the physical board is **brdBn**, where **n** is the board number starting at 1.
3. Get a count of the resources on this board using **cnf_GetResourceCount( )**. This count is a snapshot in time. The CNF_RES data structure contains information about the total number of party groups available on this board. The TSPartyGroupRes structure contains information about the number of conference resources and party resources available in a party group. The count of resources varies with the board and the media load in use on the board. Having a count of the resources on the board enables you to properly manage these resources.
4. If desired, you can specify attributes for the board using **cnf_SetBoardAttributes( )**. Attributes are contained in the TSAttribute data structure. For more information on specifying attributes, see Section 6.6, "Conferencing Attributes", on page 31.
5. Using **cnf_CreateConference( )**, create a new conference to which parties will be added. This function takes the physical board device handle returned by **cnf_Open( )** as an argument. It returns a conference handle in the TSCreateConferenceReply data structure. The conference created consumes a conference resource. A conference is bound to a party group.
6. In a multiprocessing environment where a conference was created by another process, the second process uses **cnf_AttachConference( )** to open the existing conference. For more tips and hints specific to multiprocessing, see Section 6.4, "Multiprocessing Considerations", on page 30.
7. If desired, you can specify attributes for the conference using **cnf_SetConferenceAttributes( )**. Attributes are contained in the TSAttribute data structure.
8. Using **cnf_AddParty( )**, add a party to the conference that was just created. This party can be a voice resource such as dxxxB1C1 or an IP resource such as ipmB1C1. Successfully adding a party to a conference consumes a party resource as well as a time slot on the TDM bus.
9. If desired, you can specify attributes for a party using **cnf_SetPartyAttributes( )**. Attributes are contained in the TSAttribute data structure. The default party mode is PARM_VAL_Party_Mode_NULL, which means that no time slots are assigned. To allow the party to transmit and receive in the conference, set the party mode to PARM_VAL_Party_Mode_FULL. For more information on specifying attributes, see Section 6.6, "Conferencing Attributes", on page 31.
10. Add more parties to the conference as needed. There is a limit to the number of parties that can be added to a conference in a party group (the count of resources was obtained in step 3). However, you can add parties using the conference bridging feature. For more information on bridging, see Chapter 8, "Conference Bridging".
11. Terminate your application in an orderly fashion. For example, disable events, stop listening to time slots, delete all parties, delete all conferences, close all devices, and so on.

6.2.2 Sample Scenario for Managing Resources

To better understand the programming guidelines discussed in Section 6.2.1, "Getting a Resource Count and Creating a Conference", on page 25, a sample scenario is provided. This scenario illustrates how to manage the resources on the board to create conferences and bridge conferences.

In this example, a small number of resources is used to explain the concepts. In reality, an Intel® telecom board supports many more resources.

Figure 1 represents an Intel® telecom board whose physical board device name is brdB1. It consists of two party groups, with each one containing five party resources.

Figure 1. Board brdB1 Party Groups and Party Resources

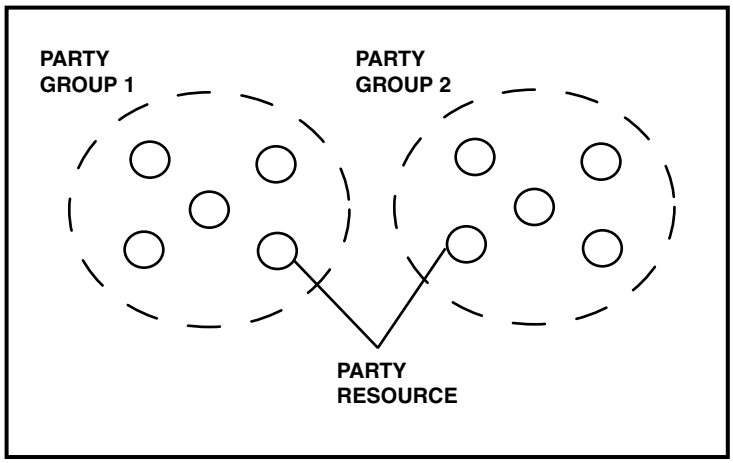

Table 1 shows resource count information that would be returned by **cnf_GetResourceCount( )** for this board.

Table 1. First Resource Count for brdB1

Type of Count	Board brdB1	Party Group 1	Party Group 2
Max number of party resources	10	5	5
Max number of conferences	4	2	2
Number of free parties	10	5	5
Number of free conferences	4	2	2

Let's assume that two conferences are needed. Figure 2 illustrates two conferences that have been created. Conference 1 in party group 1 consists of four parties. Conference 2 in party group 2 consists of two parties.

Figure 2. Conferences Created on brdB1

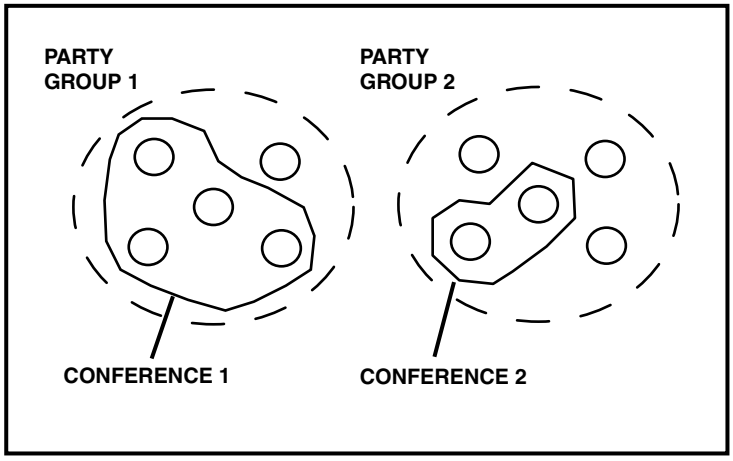

Table 2 shows resource count information that would be returned by **cnf_GetResourceCount( )** after conference 1 and conference 2 have been created on brdB1.

Table 2. Second Resource Count for brdB1

Type of Count	Board brdB1	Party Group 1	Party Group 2
Max number of party resources	10	5	5
Max number of conferences	4	2	2
Number of free parties	4	1	3
Number of free conferences	2	1	1

Next, let's assume that we want to bridge conference 1 and conference 2 so that all participants in both conferences can be linked together.

Figure 3 illustrates conference bridging between conference 1 and conference 2. One resource is required from each party group to serve as the bridge between conferences. In this example, conference 1 originally consisted of four parties with one free party. Conference 2 originally consisted of two parties with three free parties. It is good practice to have at least one free resource in a party group to allow for conferencing bridging if the need arises. This allows you to expand a conference.

Figure 3. Bridged Conferences on brdB1

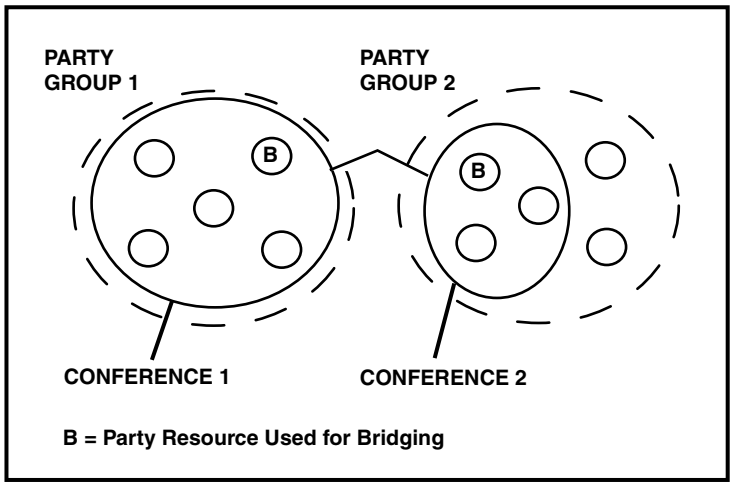

Table 3 shows resource count information that would be returned by **cnf_GetResourceCount( )** after bridging between conference 1 and conference 2 has occurred on brdB1.

Table 3. Third Resource Count for brdB1

Type of Count	Board brdB1	Party Group 1	Party Group 2
Max number of party resources	10	5	5
Max number of conferences	4	2	2
Number of free parties	2	0	2
Number of free conferences	2	1	1

6.2.3 Resource Management Rules

This section summarizes the rules and programming guidelines for resource management in a conferencing application:

- Keep track of the resource count in a party group and on a board using **cnf_GetResourceCount( )**. Each board has a predetermined number of party groups, party resources and conferencing resources based on the media load in use. Activities that cause the resource count to change include creating a conference, adding a party to a conference, bridging a conference as well as deleting a conferencing, removing a party, and deleting a bridge.
- A conference is bound to a party group; that is, a conference is created from resources in a party group. Conference resources and party resources are determined at the time the board is started in the configuration manager (DCM).
- Conference bridging consumes two time slots on the TDM bus and consumes two party resources, one from each of the conferences involved in the bridge. Conference bridging is useful for bridging across physical locations, such as between Paris and New York.

- 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, the application developer is responsible for terminating the application properly. Similarly, if an error condition abnormally terminates the application, individual conferences will not be deleted nor will individual channels be closed. In this case, either design the application to recover and manage the existing conferences or call the **cnf_DeleteAllConferences( )** function to release all conferencing resources.

6.3 Terminating an Application

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( )**
- stop listening to time slots using the appropriate function such as **dx_unlisten( )**, **gc_Unlisten( )**, and so on
- deleting all bridges if applicable using **cnf_DeleteBridge( )**
- deleting all parties using **cnf_RemoveParty( )**
- deleting all conferences using **cnf_DeleteAllConferences( )**
- closing devices using the appropriate function such as **cnf_Close( )**, **dx_close( )**, and so on.

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 4, "Event Handling" for more information about handling events.

6.4 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. Whenever conference bridging is used to bridge conferences between boards, the processes have to share certain information to create the conference bridge.

The following considerations apply when multiple processes need to create conference bridges. These considerations also 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. The name of this conference is returned in the TSCreateConferenceReply structure. Process B can then attach to that conference using **cnf_AttachConference( )**.
- 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.
- If process A creates a bridge, process A must notify process B of the existence of this bridge. Otherwise, process B cannot act on this bridge.

6.5 Multithreading Considerations

The following considerations only apply to multiple threads acting on the same board. These are in addition to multiprocessing considerations discussed in Section 6.4, "Multiprocessing Considerations", on page 30.

- Although the conferencing library allows for multiple threads to act on the same conference on the same board, you are responsible for synchronizing such calls in your application in a way that is consistent with the desired results.
- The resource counts returned by **cnf_GetResourceCount( )** 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_GetResourceCount( )**.

6.6 Conferencing Attributes

The following topics discuss attributes that can be set for a physical board, a conference, and a party:

- Overview of Conferencing Attributes
- Board Level Attributes
- Conference Level Attributes
- Party Level Attributes

6.6.1 Overview of Conferencing Attributes

Different conferencing attributes are available at the board level, conference level, and party level. The TSAttribute data structure contains information about the attributes of a physical board, conference, and party. This data structure uses several aliases: TSBoardAttribute, TSConferenceAttribute, and TSPartyAttribute. For details on this data structure, see the *Conferencing API Library Reference*.

6.6.2 Board Level Attributes

The **cnf_SetBoardAttributes( )** function sets the values of one or more attributes of a physical board. Once set, the attributes apply to all conferences and all parties on this board. For more information on when to use this function, see Section 6.2, "Managing Resources", on page 25.

These attributes include:

BOARD_Parm_Act_Talk_En (active talker feature)
: enables or disables active talker on the board. The default setting is disabled.

BOARD_Parm_ActTalkerNotifyInterval
changes the default firmware interval for active talker notification events on the physical board. The value must be passed in 10 msec units. The default setting is 20 (0.2 seconds).

BOARD_Parm_ToneClamping
enables or disables tone clamping to reduce the amount of DTMF tones heard on a per party basis on the physical board. The default setting is enabled.

6.6.3 Conference Level Attributes

The **cnf_SetConferenceAttributes( )** function sets the values of one or more attributes of a single conference. Once set, the attributes apply to all parties in the conference. For more information on when to use this function, see Section 6.2, "Managing Resources", on page 25.

These attributes include:

CONF_Parm_DTMF_Mask
specifies a mask for the DTMF digits used for volume control. The values in CNF_DIG enumeration can be ORed to form the mask. For more information about CNF_DIG values, see the CNF_VOL data structure description in the *Conferencing API Library Reference*.

CONF_Parm_ToneClamping
enables or disables DTMF tone clamping for the conference. Values are 0 to disable and 1 to enable. The default setting is enabled.

6.6.4 Party Level Attributes

The **cnf_SetPartyAttributes( )** function sets the values of one or more attributes of a single party. Once set, the attributes apply to the specific party in the conference. For more information on when to use this function, see Section 6.2, "Managing Resources", on page 25.

These attributes include:

PARTY_Parm_AGC_En
enables or disables automatic gain control. Values are 0 to disable and 1 to enable. The default setting is disabled.

PARTY_Parm_Broadcast_En
one party can speak while all other parties are muted. Values are 0 to disable and 1 to enable. The default setting is disabled.

PARTY_Parm_Coach_En
party is set to coach. Coach is heard by pupil only. Values are 0 to disable and 1 to enable. The default setting is disabled.

PARTY_Parm_Echo_Cancel
enables or disables echo cancellation for the party. Values are 0 to disable and 1 to enable. The default setting is disabled. The echo cancellation feature provides 128 taps (16 msec) of echo cancellation.

Note: This attribute is only supported by **cnf_AddParty( )** and **cnf_AddPartyByTimeslot( )**. The **cnf_SetPartyAttributes( )** function cannot be used to set or modify echo cancellation.

PARTY_Parm_Party_Mode
uses EPARTY_Parm_Party_Mode enumeration which has one of the following values:

- PARM_VAL_Party_Mode_FULL – indicates that the party may transmit and receive in the conference
- PARM_VAL_Party_Mode_NULL – indicates that no time slots are assigned. If this value is specified, no other party attributes can be specified. This is the default setting. This mode is used to reserve party resources for future use.
- PARM_VAL_Party_Mode_RCV – indicates that the party is in receive-only mode in the conference
- PARM_VAL_Party_Mode_XMIT – indicates that the party is in transmit-only mode in the conference

PARTY_Parm_Pupil_En
party is set to pupil. Pupil hears everyone including the coach. Values are 0 to disable and 1 to enable. The default setting is disabled.

PARTY_Parm_Tariff_En
party receives periodic tone for duration of the call. Values are 0 to disable and 1 to enable. The default setting is disabled.

PARTY_Parm_ToneClampingPty
DTMF tone clamping for the party. Values are 0 to disable and 1 to enable. The default setting is disabled.

6.7 Understanding User Context

User context is a mechanism that lets you match a termination event with a function call. It's a user-supplied value that is returned with the event. The user context parameter is available for each conferencing function and is used when calling the function in asynchronous mode. To retrieve the user-supplied pointer when the completion event is received, use **sr_getUserContext( )**.

For example, if you call **cnf_CreateConference( )** asynchronously in your application five times to create five conferences, each call will return an event to indicate successful completion (CNFEV_CONFCREATED) or failure (CNFEV_CREATECONFFAILED) of the function. Without the use of user context, you wouldn't know which event was associated with which conference. User context enables you to match the event with the function call by returning the user-supplied pointer originally passed to the **cnf_CreateConference( )**.

6.8 Data Structure Considerations

Take note of the following considerations 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 *Conferencing API Library Reference*.
- In synchronous mode only, when buffers are used to return an array of structures or values, if insufficient space is allocated for the buffers, the function will fail and the error

ECNF_BUFFERTOOSMALL will be returned by **ATDV_LASTERR( )**. In particular, this situation applies to the **cnf_GetActiveTalkers( )**, **cnf_GetPartyList( )**, and **cnf_GetResourceCount( )** functions. Although the function fails, the associated structure is filled with an accurate count of the elements in the array. For example, if there are five elements in the array for **cnf_GetPartyList( )**, and only two fit in the buffer, the m_nPartyCount field in TSPartyList data structure will contain 5.

6.9 Tone Detection

In certain scenarios in a voice and conferencing application, DM3 boards may detect a DTMF tone multiple times. Tone detection is performed separately on voice and conference devices. If the tone duration is long enough and the device is switched from voice to conference, then the tone may be detected twice as the algorithm detects the tone on the rising edge (for tone clamping purposes). To resolve this issue, you can add a delay before calling **dx_clrdigbuf( )**. For more information on this function, see the *Voice API Library Reference*.

Active Talker 7

This chapter provides information about invoking the active talkers feature in a conference.

Active talkers refers to parties in a conference who are providing “non-silence” energy. The conferencing software can provide indication of the active talkers in a specified conference.

The active talker feature is enabled on a physical board basis. To turn on the active talker feature, set the BOARD_Parm_Act_Talk_En value (EBoardAttributes enumeration in TSAttribute data structure) to 1 using the **cnf_SetBoardAttributes( )** function. To retrieve the list of active talkers, use **cnf_GetActiveTalkers( )**.

The frequency of active talker notification event retrieval is determined by the application by the BOARD_Parm_ActTalkerNotifyInterval value (EBoardAttributes enumeration in TSAttribute data structure) to 1 in the **cnf_SetBoardAttributes( )** function. The default setting is 20 in 10 msec units (or 0.2 seconds).Internally the active talkers are updated every 1 second.

Conference Bridging 8

This chapter provides information about bridging multiple conferences together. The following sections are included:

8.1 Conference Bridging Overview

If a conference expands beyond the number of parties permitted by the media load you have downloaded to your board, you can create a second conference to support additional conferees. The two conferences can then be connected via a conference bridge. Conference bridging is also useful for bridging across physical locations, such as between Paris and New York. Conference bridging allows all parties in two or more conferences to speak with and/or listen to one another.

The conference bridging feature uses the **cnf_CreateBridge( )** and **cnf_DeleteBridge( )** functions. These two functions allow a bridge party to be created and deleted in two existing conferences.

8.2 Creating a Conference Bridge

Follow the steps in this section to create a conference bridge. These steps cover resource management, which is a critical part of a conferencing application, as well as conference bridging. For additional programming guidelines, see Chapter 6, “Application Development Guidelines”.

1. Assuming that you are using the asynchronous programming model (the model of choice), enable a Standard Runtime Library (SRL) event handler for the conferencing device via **sr_enbhdlr( )**.
2. Open the physical board device handle using **cnf_Open( )**. The device naming convention for the physical board is **brdBn**, where **n** is the board number starting at 1.
3. Get a count of the resources on this board using **cnf_GetResourceCount( )**. The CNF_RES data structure contains information about the total number of party groups available on this board. The TSPartyGroupRes structure contains information about the number of conference resources and party resources available in a party group. The count of resources varies with the board and the media load in use on the board. Having a count of the resources on the board enables you to properly manage these resources.
4. If desired, you can specify attributes for the board using **cnf_SetBoardAttributes( )**. Attributes are contained in the TSAttribute data structure. For more information on specifying attributes, see Section 6.6, “Conferencing Attributes”, on page 31.
5. Using **cnf_CreateConference( )**, create a new conference, conference 1, to which parties will be added. This function takes the physical board device handle returned by **cnf_Open( )** as an

argument. It returns a conference handle in the TSCreateConferenceReply data structure. The conference created consumes a conference resource. A conference is bound to a party group.

Note: If the conference has already been created, use **cnf_AttachConference( )** to open the existing conference.

6. If desired, you can specify attributes for conference 1 using **cnf_SetConferenceAttributes( )**. Attributes are contained in the TSAttribute data structure.
7. Using **cnf_AddParty( )**, add a party to conference 1. This party can be a voice resource such as dxxxB1C1 or an IP resource such as ipmB1C1. Successfully adding a party to a conference consumes a party resource as well as a time slot on the TDM bus.
8. If desired, you can specify attributes for a party in conference 1 using **cnf_SetPartyAttributes( )**. Attributes are contained in the TSAttribute data structure.
9. Add more parties to the conference as needed. Based on the count of resources obtained using **cnf_GetResourceCount( )**, reserve at least one party resource free to serve as a bridge to the second conference.
10. Using **cnf_CreateConference( )**, create a new conference, Conference 2, to which parties will be added.
11. If desired, you can specify attributes for conference 2 using **cnf_SetConferenceAttributes( )**. Attributes are contained in the TSAttribute data structure.
12. Using **cnf_AddParty( )**, add a party to conference 2. This party can be a voice resource such as dxxxB1C1 or an IP resource such as ipmB1C1. Successfully adding a party to a conference consumes a party resource as well as a time slot on the TDM bus.
13. If desired, you can specify attributes for a party in conference 2 using **cnf_SetPartyAttributes( )**. Attributes are contained in the TSAttribute data structure.
14. Add more parties to the conference as needed. Based on the count of resources obtained using **cnf_GetResourceCount( )**, reserve at least one party resource free to serve as a bridge to the first conference.
15. Use **cnf_CreateBridge( )** to bridge conference 1 and conference 2. The conference device handle for each conference is used as input to this function. If a free party resource exists in each conference and two TDM bus time slots are available, this function call will be successful. If not, this function will fail.
16. Terminate your application in an orderly fashion. For example, disable events, stop listening to time slots, delete all bridges, delete all parties, delete all conferences, close all devices, and so on.

8.3 Conference Bridging Rules

Consider the following rules when using the conference bridging feature:

- Conference bridges can span multiple party groups and multiple boards.
- Each bridge that is created consumes two party resources, one from each of the conferences involved in the bridge. For an illustration, see Figure 3, “Bridged Conferences on brdB1”, on page 29.
- Each bridge that is created consumes two time slots on the TDM bus.

- 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.
- Successfully deleting a conference bridge frees one party resource from each board on which the conferences are created. If both of the conferences are created on the same board, two party resources on the same board are freed.
- If you delete a conference that is a part of a conference bridge, that conference bridge will be deleted. The resources associated with the deleted conference are then freed. The resources associated with the other conferences in the conference bridge are not released.

Volume Control 9

This chapter provides information about enabling volume control for parties 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_SetVolumeControl( )** function defines the DTMF digits used to control the volume. Digits can be defined to increase, decrease, or reset the volume. This function uses the CNF_VOL structure which specifies whether volume control is enabled or disabled and contains details on the digits used for volume control. Volume control is enabled on a physical board basis.

The **cnf_GetVolumeControl( )** 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 10

This chapter provides information on building applications using the conferencing library.

10.1 Compiling and Linking

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries
- Variables for Compiling and Linking

10.1.1 Include Files

Function prototypes and symbolic defines are determined in include files, also known as header files. Applications that use conferencing 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 conferencing library functions:

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

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

cnflib.h
Contains function prototypes and symbolic defines for the conferencing library.

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

dxxxlib.h
Contains function prototypes and symbolic defines for the voice library.

dtilib.h
Contains function prototypes and symbolic defines for the digital network interface library.

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

ipmerror.h
Contains variables for IP media library error codes.

ipmlib.h
Contains function prototypes and symbolic defines for the IP media library.

10.1.2 Required Libraries

Windows

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

libsrlmt.lib
Standard Runtime Library file. Required in all applications.

libdxxmt.lib
the primary voice library file. Required only if the application uses voice library functions directly, for example, **dx_open( )**.

libdtimt.lib
digital network interface library file. Required only if the application uses digital network interface library functions directly, for example, **dt_open( )**.

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

libipm.lib
the primary IP media library file. Required only if the application uses IP media library functions directly, for example, **ipm_Open( )**.

libcnf.lib
conferencing 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
Standard Runtime Library file. Required in all applications. Specify `-lsrl` in makefile.

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

libdti.so
digital network interface library file. Required only if the application uses digital network interface library functions directly, for example, **dt_open( )**. Specify `-ldti` in makefile.

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

libipm.so
the primary IP media library file. Required only if the application uses IP media library functions directly, for example, **ipm_Open( )**. Specify `-lipm` in makefile.

libcnf.so
conferencing 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.

10.1.3 Variables for Compiling and Linking

In System Release 6.0, the following variables have been introduced to 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 developers begin using these variables when compiling and linking applications since they will be required in future releases. 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 physical 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: 1. When used in reference to an Intel® Dialogic® analog expansion board, an audio path, or the activity happening on that audio path (for example, when you say the channel goes off-hook). 2. When used in reference to an Intel Dialogic digital expansion board, a data path, or the activity happening on that data path. 3. When used in reference to a bus, an electrical circuit carrying control information and data.

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.

computer telephony (CT): The extension of computer-based intelligence and processing over the telephone network to a telephone. Sometimes called computer-telephony integration (CTI), it lets you interact with computer databases or applications from a telephone, and enables computer-based applications to access the telephone network. Computer telephony technology supports applications such as: automatic call processing; automatic

speech recognition; text-to-speech conversion for information-on-demand; call switching and conferencing; unified messaging, which lets you access or transmit voice, fax, and e-mail messages from a single point; voice mail and voice messaging; fax systems, including fax broadcasting, fax mailboxes, fax-on-demand, and fax gateways; transaction processing, such as Audiotex and Pay-Per-Call information systems; and call centers handling a large number of agents or telephone operators for processing requests for products, services, or information.

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

conference: Ability for two or more participants in a telephone call to speak to and listen to one another in the same call.

conferencing: Ability to perform a conference.

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

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

convenience function: A class of functions that simplify application writing, sometimes by calling other, lower-level API functions.

CPE: customer premises equipment.

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.

DCM: Dialogic 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.

device: A computer peripheral or component controlled through a software device driver. An Intel 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 Intel Dialogic mediastream processing architecture, which is open, layered, and flexible, encompassing hardware as well as software components. A whole set of products from Intel are built on DM3 architecture. Contrast with Springware which is earlier-generation architecture.

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

DSP (Digital Signal Processor): A specialized microprocessor designed to perform speedy and complex operations on digital signals.

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 TDM (Time Division Multiplexing).

emulated device: A virtual device whose software interface mimics the interface of a particular physical device, such as a D/4x boards that is emulated by a D/12x or a D/xxxSC board. On a functional level, a D/12x board is perceived by an application as three D/4x boards. Contrast with physical device.

extended attribute functions: A class of functions that take one input parameter (a valid Intel Dialogic device handle) 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.

physical device: A device that is an actual piece of hardware, such as a D/4x board; not an emulated device. See emulated device.

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

resource: Functionality (for example, voice-store-and-forward) 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.

route: Assign a resource to a time slot.

Springware: Software algorithms build into the downloadable firmware that provides the voice processing features available on all Intel voice boards. The term Springware is also used to refer to a whole set of boards from Intel built using this architecture. Contrast with DM3 which is newer-generation architecture.

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 Intel telecom devices are contained in the SRL. See standard runtime library (SRL).

standard runtime library (SRL): An Intel Dialogic software resource containing event management and standard attribute functions and data structures used by all Intel telecom 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.

system release: The software and user documentation provided by Intel that is required to develop applications.

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**.

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.

time slot assignment: The ability to route the digital information contained in a time slot to a specific analog or digital channel on an expansion board. See also device channel.

Index

M

P

R

S

T

U

V