Audio Conferencing API for Linux Operating Systems

Programming Guide

September 2002

05-1879-001

Publication Date: September 2002

Document Number: 05-1879-001

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 Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices** and other contact information, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Contents

Figures

Tables

About This Publication

The following topics provide information about this publication:

Purpose

This publication provides guidelines for building computer telephony applications using the audio conferencing API.

This publication is a companion guide to the *Audio Conferencing API Library Reference* that provides details on functions and parameters in the audio conferencing library.

Intended Audience

This publication is written for the following audience:

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

How to Use This Publication

Refer to this publication after you have installed the hardware and the system software which includes the audio conferencing software.

This publication assumes that you are familiar with the Linux* operating system and the C programming language.

The information in this guide is organized as follows:

- Chapter 1, "Product Description" introduces the key features of the audio conferencing library and provides a brief description of each feature.

- 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 and how to open/close devices.
- Chapter 4, “Event Handling” provides information about functions used to handle events.
- Chapter 5, “Error Handling” provides information about handling errors in your application.
- Chapter 6, “Application Development Guidelines” provides guidelines for developing applications with the audio conferencing library.
- Chapter 7, “Active Talker” provides details about the active talker feature.
- Chapter 8, “Conference Bridging” discusses how to connect conferences together via a conference bridge.
- Chapter 9, “Volume Control” explains how to enable volume control for conferees.
- Chapter 10, “Building Applications” discusses compiling and linking requirements such as include files and library files.

Related Information

Refer to the following documents and Web sites for more information on developing your application:

- *Audio Conferencing API Library Reference*
- *Digital Network Interface Software Reference*
- *MSI/SC Software Reference*
- *Standard Runtime Library API Library Reference*
- *Standard Runtime Library API Programming Guide*
- *Intel® NetStructure™ on DM3™ Architecture Configuration Guide*
- *System Release Guide*
- *System Release Update*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/network/csp/* (for product information)

Product Description 1

This chapter provides information about the features available in the audio conferencing library.

Audio conferencing is software that supports development of host-based audio conferencing applications on Intel® NetStructure™ on DM3 architecture boards.

Note: In order to enable audio conferencing on an Intel® NetStructure board, you must configure the board with a Media Load that supports conferencing resources. For complete information about Media Loads, refer to the *Intel ®NetStructure™ on DM3™ Architecture Configuration Guide*.

Key features of the audio conferencing software include:

- Bridging multiple conferences together so that all conferees in two or more established conferences can speak with and/or listen to one another.
- Coach/pupil feature allows two selected conferees to establish a private subconference with respect to the overall conference.
- DTMF digit detection for any conferee, allowing the application to determine when or if any party has generated a DTMF digit.

  Note: Any conferee in receive-only mode cannot generate DTMF digits within the conference, therefore any digits dialed by a conferee in receive-only mode will not generate DCBEV_DIGIT events.
- Volume control for any conferee by issuing pre-programmed DTMF digits.
- Tone clamping that allows each conferee to reduce the amount of DTMF tones heard during a conference.

  Note: DTMF tones may be heard by conferees if the application encourages the user to repeatedly press DTMF tones, for example, press 9 to raise volume.
- Automatic Gain Control (AGC) for all conferees.
- Active talker indication to determine which conferees in any given conference are currently talking. The active talker feature can be set to indicate which conferees are talking the loudest or talking for the longest amount of time.
- Monitoring feature enabling many participants to monitor a single conference without interrupting the conference.
- Echo cancellation for each talker.

Programming Models 2

This chapter briefly discusses the Standard Runtime Library and supported programming models. The following topics are discussed:

2.1 Standard Runtime Library

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

When developing audio conferencing applications, refer to the Standard Runtime Library documentation in tandem with the audio conferencing library documentation. For more information about the Standard Runtime Library, see the *Standard Runtime Library API Library Reference* and the *Standard Runtime Library API Programming Guide*.

2.2 Asynchronous Programming Model

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

For more information about asynchronous programming models, see the *Standard Runtime Library API Library Reference*.

2.3 Synchronous Programming Model

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

For more information about asynchronous programming models, see the *Standard Runtime Library API Library Reference*.

Device Handling 3

This chapter discusses the following topics related to device handling on physical boards:.

3.1 Key Concepts

The concepts discussed in this section are key to understanding devices and device handling.

device
: A device is a computer peripheral or component controlled through a software device driver. A resource board, such as a voice resource, conferencing resource, and network interface board contain one or more logical board devices. Each digital signal processor (DSP) or time slot on the board is considered a device. The audio conferencing API distinguishes between a board device and a DSP device.

device name
: A device name is 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 type of device to be opened. The “xx” prefix refers to “dcb” for conferencing device, ”dti” for network interface device, “dxxx” for voice device and so on.

device handle
: A device handle is 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 “xx” prefix refers to “dcb” for conferencing device, ”dti” for network interface device, “dxxx” for voice device and so on. The device handle is used for all operations on that device.

3.2 Device Names

The system software assigns device names in a process described in the following topics:

- Overview of Device Names
- Designating Device Types
- Device Naming Rule
- Rules for DM3 Architecture Boards

3.2.1 Overview of Device Names

The system software creates standard device names for conferencing boards and DSPs that reside on conferencing boards. These names are input as the **name** parameter to, for example, the

dcb_open( ) function, which return the device handles necessary for many essential API calls, such as **dcb_addtoconf( )** and **dcb_getbrdparm( )**.

3.2.2 Designating Device Types

The system software designates devices as the following types:

- **Audio conferencing**. Device names for this type receive the prefix **dcb**.
- **Digital network interface**. Device names for this type receive the prefix **dti.**
- **Voice**. Device names for this type receive the prefix **dxxx**.

For more information on device types for each technology, see the appropriate documentation, such as the Digital Network interface API and Voice API documentation sets.

3.2.3 Device Naming Rule

A device name is assigned to each conference device or each component in a board as follows:

dcbBn
where **n** is the device number assigned in sequential order down the list of sorted conference boards.

A conference device name can be appended with a DSP identifier:

dcbBnDy
where **y** corresponds to one of the board's DSPs. Examples of DSP device names for conferencing boards are **dcbB1D1** and **dcbB1D2.** Individual conferences are associated with DSPs and may not span DSPs. Each conferencing board DSP has a certain number of conferencing resources, depending on the board model and the Media Load that is used to configure the board. Refer to the *Intel® NetStructure™ on DM3™ Architecture Configuration Guide* for information about the number of conference resources supported by DM3 boards. Every time a conferee or a conference bridge is added to a conference, a conference resource is used. When a conferee is removed from a conference and/or a conference bridge is deleted, the associated resource is freed. If an entire conference is deleted, all associated resources are freed. Refer to Section 6.4, "Resource Allocation", on page 22 for more information about conferencing resource usage.

3.2.4 Rules for DM3 Architecture Boards

When using the audio conferencing API, the following rules apply specifically to DM3 board naming and numbering:

- A single physical DM3 board device can contain multiple virtual boards that are each numbered in sequential order; for example, a DM/V960A-4T1 board with four digital network interfaces contains four virtual network interface boards that would follow a sequential numbering pattern such as dtiB1, dtiB2, dtiB3, dtiB4.
- All DM3 board devices are numbered in sequential order based on the logical ID assigned by the DM3 driver: the board having the lowest logical ID will be assigned the next board number, and so on.

3.3 Opening and Using Devices

When you open a file, it returns a unique file descriptor for that file. The following is an example of a file descriptor:

```
int file_descriptor;
file_descriptor = open(filename, mode);
```

Any subsequent action you wish to perform on that file is accomplished by identifying the file using **file_descriptor**. No action can be performed on the file until it is first opened.

Conferencing boards and DSPs work in a similar manner. You must first open a conference device using **dcb_open( )** before you can perform any operation on it.

When you open a board or DSP device connected to the time division multiplexing (TDM) bus using **dcb_open( )**, the value returned is a unique device handle for that particular open process. Typically, the device handle is referred to as **devh**:

```
int dspdevh; /*indicates device handler for DSP device*/
dspdevh = dcb_open("dcbBnDy", 0)
```

The DSP device name is **dcbBnDy** where B is followed by the board number and D is followed by the number of the DSP. An example is dcbB1D2 for board 1, DSP 2.

The device handle for a digital network interface device is referred to as **dtih** (see the *Digital Network Interface Software Reference* for details):

```
int dtih;
dtih = dt_open(dtiBxTx,mode)
```

The device name is **dtiBxTx** where B is followed by the unique board number and T is followed by the number of the time slot (digital channel), 1 to 24 for T-1 or 1 to 30 for E-1.

For more information on device naming, see Section 3.2, "Device Names", on page 13.

To use an audio conferencing library function on a DSP, you must identify the DSP with its DSP device handle, **devh**. The DSP device name is used only when opening a DSP device, and all actions after opening must use the handle **devh**.

Board devices are opened by following the same procedure, where **devh** refers to the board device handle.

Note: Boards and DSPs are considered separate devices. It is possible to open and use a DSP without ever opening the board it is on. There is no board-DSP hierarchy imposed by the driver.

In applications that spawn child processes from a parent process, device handles are not inheritable from the parent process to the child process. Make sure that devices are opened in the child process.

The system software provides libraries of C language functions that enable you to control the boards and DSPs. For details on opening and closing channels and boards, refer to the function reference descriptions for:

- **dt_open( )** and **dt_close( )** in the *Digital Network Interface Software Reference*

- **dcb_open( )** and **dcb_close( )** in the *Audio Conferencing API Library Reference*

Event Handling 4

This chapter provides information on functions used to retrieve and handle events that are generated by the functions in the audio conferencing library. Topics include:

4.1 Overview of Event Handling

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

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

4.2 Event Management Functions

Event management functions are used to retrieve and handle events being sent to the application from the firmware. These functions are contained in the Standard Runtime Library and defined in *srllib.h*. The library provides a set of common system functions that are device independent and are applicable to all devices. For more information on event management and event handling, see the *Standard Runtime Library API Programming Guide*.

To enable an event handler for a specified event, follow these steps:

1. Call **sr_enbhdlr( )**. This function specifies the event and the application defined event handler that is called from a signal handler.
2. Call **dcb_setdigitmsk( )** or **dcb_evtstatus( )** functions. These functions set the digit message mask.

Note: The request for an event to be posted to an event handler must be specified using both the **sr_enbhdlr( )** and either the **dcb_setdigitmsk( )** or **dcb_evtstatus( )** functions.

Each of the Event Management functions applicable to conference devices are listed in the following tables. **Table 1** lists values that are required by event management functions. **Table 2** lists values that are returned for event management functions that are used with conferencing devices.

Table 1. Conference Device Inputs for Event Management Functions

Event Management Function	Conference Device specific Input	Value
sr_enbhdlr( ) Enable event handler	evt_type	DCBEV_CTU DCBEV_DIGIT
sr_dishdlr( ) Disable event handler	evt_type	DCBEV_CTU DCBEV_DIGIT.
sr_waitevt( ) Wait for next event		N/A
sr_waitevtEx( ) Wait for next event		N/A

Table 2. Conference Device Returns from Event Management Functions

Event Management Function	Conference Device specific Input	Value
sr_getevtdev( ) Get device handle	device	Conference device handle.
sr_getevttype( ) Get event type	event type	DCBEV_DIGIT DCBEV_CTU
sr_getevtlen( ) Get event length	event length	Number of bytes in the data returned.
sr_getevtdatap( ) Get pointer to event data	event data	Pointer to DCB_DIGITS structure for DCBEV_DIGIT. Pointer to DCB_CT structure, the updated resource table for DCBEV_CTU.

Error Handling 5

This chapter discusses how to handle errors that can occur when running an audio conferencing application.

All the audio conferencing library functions return a value that indicates the success or failure of the function call. Audio conferencing functions return one of the following values:

0
: function success

-1
: function error

Any call to a library function should therefore check for a return value indicating an error. This can be done using a format similar to the following:

```
/* call to Audio Conferencing library function */

if (dcb_xxx(arguments) == -1) {
          /* error handling routine */
}
/* successful function call -
          continue processing ...  */
```

Using this technique ensures that all errors resulting from a library call will be trapped and handled properly by the application.

If an audio conferencing library function fails, call the Standard Runtime Library standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to determine the reason for failure. For more information about these functions, see the *Standard Runtime Library API Library Reference*.

Notes: ***1.*** The function **dcb_open( )** is an exception to the error-handling rules. A **dcb_open( )** function call returns a device handle if the function call is successful. A device handle is a non-zero value. If **dcb_open( )** fails, the return code is -1 and the specific error is found in **errno** defined in *errno.h*.

2. If the error returned by **ATDV_LASTERR( )** is **EDT_SYSTEM**, a Windows system error has occurred, and you need to check the global variable **errno** defined in *errno.h*.

For a list of errors that can be returned by an audio conferencing library function, see the *Audio Conferencing API Library Reference*.

Application Development Guidelines 6

This chapter provides programming guidelines and techniques for developing an application using the audio 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 *dcblib.h*, *dtilib.h*, *msilib.h* and *srllib.h* files.

6.2 Initialization of DM3 Board Parameters

As a first step, an audio conferencing application must initialize board-level parameters. Use **dcb_setbrdparm( )** to set the board-level parameters Specific setting choices for boards that use DM3™ Architecture include:

MSG_ACTID (Active Talker Feature)
: Indicates Active Talker feature status (enabled or disabled). Possible values are ACTID_ON or ACTID_OFF. ACTID_OFF is the default.

MSG_ALGORITHM (Active Talker Algorithm)
: Determines which algorithm is used to designate active talkers. Active talkers can be determined by who is talking for the longest amount of time or who is talking the loudest. Possible values are ALGO_LONG or ALGO_LOUD.

MSG_VOLDIG (Volume Control Digits)
: Defines the volume control status and volume up/down/reset digits as defined in the MS_VOL data structure.

MSG_TONECLAMP (Tone Clamp Activation)
: Enables tone clamping on a per party basis to reduce the amount of DTMF tones heard in a conference. Possible values are TONECLAMP_ON or TONECLAMP_OFF. TONECLAMP_OFF is the default.

MSG_ACTTALKERNOTIFYINTERVAL (Active Talker Notification Event Interval)
Changes the default firmware interval for Active Talker Notification events. The value must be passed in 100ms units.

If conference resource table updates are desired by the application, use the **dcb_evtstatus( )** to enable event generation.

6.3 Terminating

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
- Stop listening to time slots
- Deleting all conferences
- Closing devices

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 Resource Allocation

The available conference resource count is retrieved using the **dcb_dsprescount( )** function. The number of available conference resources depends on which Media Load is downloaded to your board. Refer to the *Intel® NetStructure™ on DM3™ Architecture Configuration Guide* for information about Media Loads and their supported conference resources. Calling any of the following functions will cause the available resource count to change:

dcb_addtoconf( )
uses one resource every time a conferee is successfully added to a conference

dcb_CreateBridge( )
uses one resource within the master conference for each conference that is bridged to the master conference, uses only one resource in each individual conference that is bridged to the master conference.

dcb_DeleteAllConferences( )
frees all of a conference board's resources

dcb_delconf( )
frees all resources in use by the conference, including the monitor

dcb_DeleteBridge( )
frees all resources used by the conference bridge

dcb_estconf( )
uses the total number of conferees established in the new conference

dcb_monconf( )
uses one resource

dcb_remfromconf( )
frees one resource

dcb_unmonconf( )
frees one resource

Active Talker 7

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

Active talkers are those conferees providing “non-silence” energy. The conferencing board can provide indication of the active talkers in a specified conference.

To retrieve active talkers, turn on the active talker feature by setting MSG_ACTID to ACTID_ON using the **dcb_setbrdparm( )** function. Then, retrieve active talkers using the **dcb_gettalkers( )** function. The frequency of active talker retrieval is determined by the application using the MSG_ACTALKERNOTIFYINTERVAL board-level parameter, although internally the active talkers are updated every 100 ms.

The internal implementation of the active talker feature consists of three parts:

1. The DSPs on the conferencing boards are programmed to furnish non-silence information for each of the conference resources every 100 ms. This information is in bit form and is referred to as *active talker indicator (ATI) bits*. If the bit associated with a resource is on, it indicates non-silence energy in that resource during the time interval. The ATI bit pattern for a specific instant is retrieved using the **dcb_GetAtiBitsEx( )** function.
2. The resource assignment table contains DSP-to-conferees resource assignments for established conferences.
3. A combination of parts one and two is used to provide the output when the **dcb_gettalkers( )** function is called.

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 number of conferees permitted by the Media Load you have downloaded to your board, you can create a second conference to support the additional conferees and connect the two conferences via a conference bridge. Conference bridging allows all conferees in two or more conferences to speak with and/or listen to one another.

The conference bridging feature uses the **dcb_CreateBridge( )** and **dcb_DeleteBridge( )** functions. These two functions allow for a bridge party to be created and deleted in two separate conferences. The **dcb_CreateBridge( )** function translates to adding a full duplex party on each conference. The bridge parties have no attributes (NULL) and simply act as conduits between the two conferences.

The conference bridging feature uses TS_BRIDGECDT data structure to provide information about the conference bridge. This structure is composed of three elements; two MS_CDT structures and an unsigned integer. The two MS_CDT structures are filled by the library as part of the **dcb_CreateBridge( )** function and returned back to the application. The unsigned integer provides the application with a unique bridge identification number.

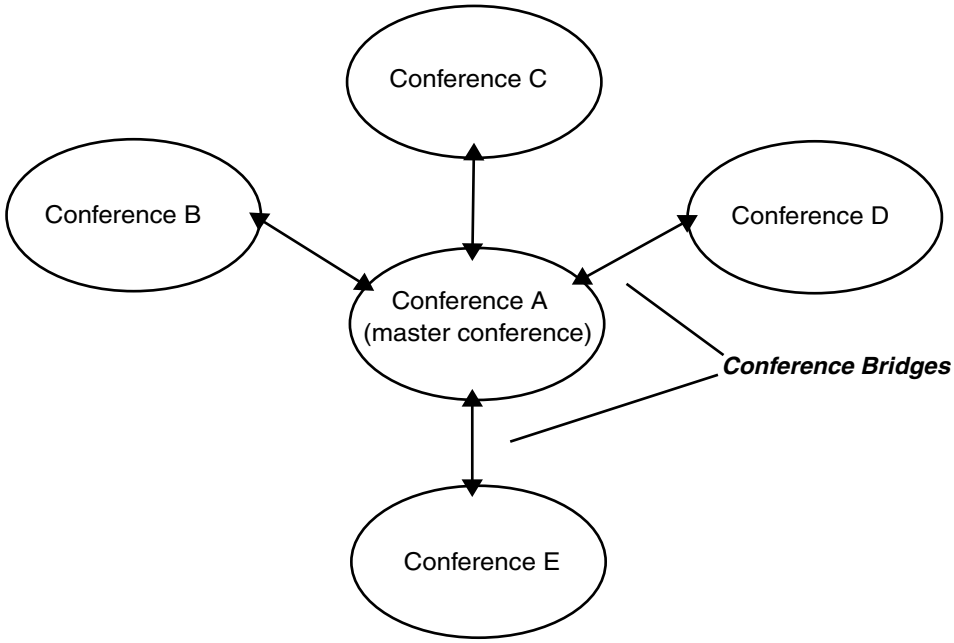
The MS_CDT data structures of the bridged conferences are returned as part of the TS_BRIDGECDT data structure. This structure makes it easy for application to track information about the conferences.

Note: 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 conference bridging feature is implemented using a star configuration. The **dcb_CreateBridge( )** function designates a conference as the master conference and then adds all other conferences to the master conference as though they were individual NULL conferees.

For example, to connect five separate conferences (Conference A, Conference B, Conference C, Conference D and Conference E) you would have to invoke the **dcb_CreateBridge( )** conference four times, adding a different conference to the master conference (Conference A) each time. Figure 1 shows the star configuration of the conference bridging feature.

Figure 1. Star Configuration for Conference Bridging



8.2 Conference Bridging Limitations

The following limitations must be considered when using the conference bridging feature:

- Each bridge that is created consumes one conferencing resource in the master conference and one conference resource in the conference that is connected to the master conference.
- Conference bridges can span multiple DSPs and multiple conferencing boards.
- Conferences cannot be simultaneously bridged to multiple master conferences.
- You cannot bridge one master conference to another master conference.
- The coach/pupil feature does not span conference bridges. Coach and pupil must be in the same conference.

Volume Control 9

This chapter includes information about enabling volume control for conferees.

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

The MSG_VOLDIG parameter in **dcb_setbrdparm( )** allows the application to define the digits that cause the volume level to be adjusted up, down or back to the default value. When a conferee other than a monitor or receive-only party presses a digit programmed to change volume, the received signal is adjusted.

Note: The DTMF digits dedicated to volume control do not cause events to be sent to the application, even if digit detection is turned on using the **dcb_setdigitmsk( )** function. Volume control digits are used only for controlling volume.

Building Applications 10

This chapter provides information on building applications using the audio conferencing library. The following topics are discussed:

10.1 Compiling and Linking

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

10.1.1 Include Files

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

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

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

msilib.h
Contains function prototypes and symbolic defines for the Modular Station Interface library.

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

errno.h
Contains variables for error codes.

10.1.2 Required Libraries

You must link the following library files in the order shown when compiling your audio conferencing application:

libsrlmt.so
Standard Runtime Library file. Specify **-lsrl** in makefile.

libdtimt.so
Digital Network Interface library file. Specify **-ldti** in makefile.

10.2 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} -lgc
```

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.

Index

M

P

R

S

T

V