Audio Conferencing API for Linux and Windows Operating Systems

Library Reference

May 2006

05-1843-004

Publication Date: May 2006

Document Number: 05-1843-004

Intel
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 and Compute Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:
http://www.intel.com/buy/networking/telecom.htm

Contents

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1843-004	April 2006	This document version was released in conjunction with SR 6.1 Windows* for CompactPCI* GA. dcb_GetPartyParm( ), and dcb_SetPartyParm( ): Added functions to support direct application control of output volume (per party). Also added cross references to these functions. Function Information: Added note to the following functions indicating that they are **not** supported on HMP software: dcb_GetAtiBitsEx( ), dcb_GetPartyParm( ), and dcb_SetPartyParm( ). Error Codes: Added HMP-only error code (EDT_HSIBRIDGEERR). Also added to dcb_addtoconf( ) and dcb_estconf( ) functions. dcb_setbrdparm( ) and dcb_getbrdparm( ): Changed the description of the MSG_ACTID parameter to indicate that it enables/disables active talker identification (or notification) and not the active talker feature itself (IPY00006584 = PTR 36199 and IPY00010946 = PTR 36323).
05-1843-003	February 2005	dcb_GetAtiBitsEx( ) and dcb_gettalkers( ) functions: Described the functions and the active talker feature more accurately (PTR 34210). dcb_getbrdparm( ) and dcb_setbrdparm( ) functions: Described the active talker parameters more accurately (PTR 34210), including stating that by default it is enabled for DM3 architecture boards, removing the MSG_ALGORITHM parameter (PTR 34382), which does not apply to DM3, and clarifying the MSG_ACTTALKERNOTIFYINTERVAL parameter, providing default value, units, etc. Also, clarified the MSG_TONECLAMP parameter (PTR 34672). dcb_CreateBridge( ), dcb_dsprescount( ), and dcb_estconf( ) functions: . Added details on maximum conference size, maximum number of conference resources, and conference bridging and clarified descriptions. dcb_remfromconf( ) function: Added note to **cdt** parameter that the party attributes are ignored. MS_CDT data structure: Added details and clarified description of structure and attributes. Added note that MSPA_PARTYTONECLAMP attribute is used to enable tone clamping on a per-party basis. Added note on using MSPA_MODEXMITONLY to add a music resource to a conference in a background music application, such as a dating chat line application where two callers talk while music plays in the background.

Document No.	Publication Date	Description of Revisions
05-1843-002	August 2004	Global changes: Removed references to operating system errno global variable and errno.h from example code and include files lists (PTR 28014). Error Codes chapter: For EDT_SYSTEM, corrected error code description to say "operating system error." Added E_MSSYSTEM to list of codes. MS_CDT data structure: Use echo cancellation on analog lines, especially with Coach/Pupil (PTR 32072). Enable echo cancellation using dcb_estconf( ) or dcb_addtoconf( ); you cannot use dcb_setcde( ) to enable or disable echo cancellation (PTR 29542). dcb_setcde( ) function: Cannot use dcb_setcde( ) to enable or disable echo cancellation (PTR 29542). dcb_estconf( ) function: Maximum value for numpty is 24. (PTR 31432)
05-1843-001	September 2002	Initial version of document. Much of the information contained in this document was previously published in the *Audio Conferencing Software Reference for Linux*, document number 05-0510-004 and the *Dialogic Audio Conferencing Software Reference for Windows*, document number 05-0512-002.

About This Publication

The following topics provide information about this publication:

Purpose

This publication provides a reference to all functions, parameters and data structures in the audio conferencing library. These functions and parameters are used to build audio conferencing applications.

This publication is a companion document to the *Audio Conferencing API Programming Guide,* the *Standard Runtime Library API Programming Guide* and the *Standard Runtime Library API Library Reference*.

Intended Audience

This information is intended for:

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

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 publication is organized as follows:

- Chapter 1, “Function Summary by Category” introduces the various categories of audio conferencing API functions and provides a brief description of each function.

- Chapter 2, "Function Information" provides an alphabetical reference to all audio conferencing API functions.
- Chapter 3, "Events" includes an alphabetical reference to events that may be returned by the audio conferencing library.
- Chapter 4, "Data Structures" provides an alphabetical reference to all data structures used by the audio conferencing library.
- Chapter 5, "Error Codes" presents a listing of error codes that may be returned by the audio conferencing library.

Related Information

See the following documents and Web sites for more information:

- *Audio Conferencing API Programming Guide*
- *Standard Runtime Library API Library Reference*
- *Standard Runtime Library API Programming Guide*
- *Digital Network Interface Software Reference*
- *MSI/SC Software Reference*
- *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)

Function Summary by Category 1

The audio conferencing library functions provide the necessary building blocks to create conferencing applications with Intel NetStructure® on DM3 architecture boards. These functions can be divided into the following categories:

1.1 Auxiliary Functions

The following functions add flexibility while using the active talker feature and allow your application to get/set the status of audio conferencing API events:

dcb_evtstatus( )
gets or sets the status of a process-wide event, not a specific device handle

dcb_GetAtiBitsEx( )
gets active talkers for all conferences on a DSP

dcb_gettalkers( )
gets active talkers and party attributes for a specific conference

1.2 Conference Management Functions

The following functions are used to manage all conference activities:

dcb_addtoconf( )
adds one conferee to an existing conference

dcb_CreateBridge( )
creates a conference bridge

dcb_delconf( )
deletes an individual conference

dcb_DeleteAllConferences( )
deletes all established conferences

dcb_DeleteBridge( )
deletes a conference bridge

dcb_estconf( )
establishes a conference

dcb_getcde( )
retrieves the attributes of a conference (conference descriptor element)

dcb_getcnflist( )
gets a list of conferees in a conference

dcb_GetPartyParm( )
gets current parameters for a conferee

dcb_monconf( )
adds a monitor to a conference

dcb_remfromconf( )
removes a conferee from a conference

dcb_setcde( )
sets the attributes of a conferee

dcb_SetPartyParm( )
gets current parameters for a conferee

dcb_unmonconf( )
removes the monitor from a conference

1.3 Configuration Functions

These functions set the conference board or digital signal processor (DSP) device parameters, check the status of the conference board or DSP device parameter settings, and retrieve or set specific information about DSPs or conferences. The following configuration functions exist in the audio conferencing library:

dcb_dsprescount( )
retrieves the free conferencing resource count

dcb_getbrdparm( )
gets conference board device parameters

dcb_getdigitmsk( )
reads the digit event message mask

dcb_setbrdparm( )
changes conference board device parameters

dcb_setdigitmsk( )
sets the digit event message mask

1.4 Device Management Functions

These functions are used to open and close conference devices. A conference device can be a board or an individual DSP on a board. Before using any audio conferencing library functions, a conference device must be opened. Each time a device is successfully opened using **dcb_open( )**, the function returns a unique device handle.

dcb_open( )
opens a conference device

dcb_close( )
closes a conference device

Function Information 2

This chapter provides an alphabetical reference to the functions in the audio conferencing API.

2.1 Function Syntax Conventions

The audio conferencing functions use the following syntax:

```
int dcb_function(devh, parameter1, ...parameterN)
```

where:

int
: refers to the data type integer

dcb_function
: represents the function name. All conferencing functions begin with “dcb”

devh
: represents the device handle, which is a numerical reference to a conference device. A device handle is obtained when a conference device is opened, it is used for all operations on that device.

parameter1
: represents the first parameter

parameterN
: represents the last parameter

dcb_addtoconf( )

Name: int dcb_addtoconf(devh, confid, cdt)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
MS_CDT *cdt	• pointer to conference descriptor element

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_addtoconf( )** function adds one conferee to an existing conference.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference to which the conferee will be added
cdt	points to an MS_CDT data structure that defines the attributes of the added conferee

Notes: ***1.*** Only one conferee can be added at a time using this function.

2. Invoking this function uses one conferencing resource each time a conferee is successfully added to a conference.

When a conferee is added to a conference, the TDM bus time slot number to listen to is returned in the chan_lts field of the MS_CDT data structure. The chan_attr field in the MS_CDT structure is redefined in the *msilib.h* file as follows:

```
#define   chan_lts     chan_attr
```

The chan_lts value must be used by the application to listen to the conferenced signal.

Cautions

This function fails when:

- The device handle specified is invalid.
- Too many parties are specified for a single conference.
- The conference identifier is invalid.
- Conference resources are not available on the DSP.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

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

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{
   int  dspdevh;                        /* Conference board DSP device descriptor   */
   int  tsdevh1, tsdevh2, tsdevh3;      /* Time slot device descriptor */
   MS_CDT  cdt[NUM_PARTIES];            /* Conference descriptor table */
   int  confid;                         /* Conference identifier */
   SC_TSINFO tsinfo;                    /* Time slot information structure */
   long  scts;                          /* TDM bus time slot  */

   /* Open conference board 1, DSP 1 device */
   if ((dspdevh = dcb_open("dcbB1D1",0)) == -1) {
      printf( "Cannot open dcbB1D1: system error/n");
      exit(1);
   }

   /* Assume conference board is connected to a DTI via TDM bus. */

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Fill in the time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;
```

```
/* Get the TDM bus transmit time slot of tsdevh1 */
if (dt_getxmitslot(tsdevh1, &tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Set up CDT structure */
cdt[0].chan_num  = (int)scts ;         /* TDM bus transmit time slot.... */
cdt[0].chan_sel  = MSPN_TS ;           /* ...returned from dt_getxmitslot() */
cdt[0].chan_attr = MSPA_NULL;          /* Conferee has no special attributes */

/* Open DTI board 1, time slot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf( "Cannot open dtiB1T2: system error/n");
   exit(1);
}

/* Get the TDM bus transmit time slot of tsdevh2 */
if  (dt_getxmitslot(tsdevh2, &tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* TDM bus time slot to be conferenced */
cdt[1].chan_num  = (int)scts;         /* TDM bus time slot returned... */
cdt[1].chan_sel  = MSPN_TS;           /* ...from dt_getxmitslot() */
cdt[1].chan.attr = MSPA_NULL;         /*Conferee has no special attributes*/

/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Open DTI board 1, time slot 3 */
if ((tsdevh3 = dt_open("dtiB1T3",0)) == -1) {
   printf("Cannot open dtiB1T3: system error/n");
   exit(1);
}

/* Do a listen for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if  (dt_listen(tsdevh2,&tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Fill in the time slot information structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;

/* Get the TDM bus  transmit time slot of tsdevh3 */
if  (dt_getxmitslot(tsdevh3, &tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh3));
   exit(1);
}
```

```
   /* Add another conferee to conference */
   cdt[0].chan_num  = (int)scts;    /* scts is the time slot... */
   cdt[0].chan_sel  = MSPN_TS;      /* ...returned from getxmitslot()*/
   cdt[0].chan_attr = MSPA_COACH;

   if (dcb_addtoconf(dspdevh, confid, &cdt) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }
   else
      printf("Party added to conference\n");

   /* Do a listen for tsdevh3 */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

   if (dt_listen(tsdevh3,&tsinfo) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh3));
      exit(1);
   }

   /* Continue processing */

   /* Unlisten the time slots */
   if (dt_unlisten(tsdevh1) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   if (dt_unlisten(tsdevh2) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   if (dt_unlisten(tsdevh3) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh3));
      exit(1);
   }

   /* Delete the conference */
   if (dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d. Error Message = %s", confid,
             ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   if (dt_close(tsdevh1) == -1) {
      printf("Error closing tsdevh1\n");
      exit(1);
   }

   if (dt_close(tsdevh2) == -1) {
      printf("Error closing tsdevh2\n");
      exit(1);
   }

if  (dt_close(tsdevh3) == -1) {
      printf("Error closing tsdevh3\n");
      exit(1);
   }

   if (dcb_close(dspdevh) == -1){
      printf("Cannot close dcbB1D1: system error/n");
      exit(1);
   }
}
```

See Also

- **dcb_delconf( )**
- **dcb_DeleteAllConferences( )**
- **dcb_estconf( )**
- **dcb_remfromconf( )**

dcb_close( )

Name: int dcb_close(devh)

Inputs: int devh
- valid board or DSP device handle

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Device Management

Mode: synchronous

Platform: DM3

Description

The **dcb_close( )** function closes the conference device previously opened by **dcb_open( )**. The devices are either conference boards or individual DSPs on the conference board. The **dcb_close( )** function releases the device handle. Refer to the *Audio Conferencing API Programming Guide* for complete information about device names.

Parameter	Description
devh	specifies the valid device handle obtained when the board or DSP device was opened using **dcb_open( )**

Cautions

- This function fails if the device handle is invalid.
- The **dcb_close( )** function affects only the link between the calling process or thread and the device. Other processes or threads are unaffected by **dcb_close( )**.
- If event notification is active for the device to be closed, call the Standard Runtime Library **sr_dishdlr( )** function to disable the event handler prior to calling **dcb_close( )**.
- A call to **dcb_close( )** does not affect the configuration of the conferencing board.

Errors

The **dcb_close( )** function does not return errors in the standard return code format. If an error occurred during the **dcb_close( )** call, a -1 will be returned, and it indicates a operating system error.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

main()
{
   int dspdevh;             /* DSP device descriptor variable */
   /* Open conference board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open DSP dcbB1D2: system error/n");
      exit(1);
   }

   /*  Continue Processing  */

   /* Done processing - close device */
   if (dcb_close(dspdevh) == -1) {
      printf("Cannot close DSP dcbB1D2: system error/n");
      exit(1);
   }
}
```

See Also

- **dcb_open( )**

dcb_CreateBridge( )

Name: int dcb_CreateBridge(hSrlDeviceA, nConferenceIDA, hSrlDeviceB, nConferenceIDB, Bridgecdt, unMode, rfu)

Inputs:

int hSrlDeviceA	• valid DSP device handle of the master conference
int nConferenceIDA	• conference identifier for the master conference
int hSrlDeviceB	• valid DSP device handle of the conference that will be bridged to the master conference
int nConferenceIDB	• conference identifier of the conference that will be bridged to the master conference
TS_BRIDGECDT * Bridgecdt	• pointer to a conference bridge descriptor element
unsigned short unMode	• mode of the function
void *rfu	• void pointer that is reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous or asynchronous

Platform: DM3

Description

The **dcb_CreateBridge( )** function establishes a bridge between two conferences. A bridge is a link between conferences that allows each conferee within the distinct conferences to communicate as though they were part of a single conference. A bridge connects two conferences together by adding a secondary conference to a master conference as though it were an individual NULL conferee. A bridge consumes two conference resources (one in each conference).

Parameter	Description
hSrlDeviceA	specifies the valid device handle for the master conference obtained when the DSP device was opened using **dcb_open( )**
nConferenceIDA	indicates the conference identifier number of the master conference
hSrlDeviceB	specifies the valid DSP device handle used by the conference that will be bridged to the master conference
nConferenceIDB	indicates the conference identifier number of the conference that will be bridged to the master conference

Parameter	Description
Bridgecdt	points to the TS_BRIDGECDT data structure that defines the attributes of the conference bridge
unMode	specifies whether the function should run asynchronously or synchronously. Possible values are as follows: • EV_ASYNC – Function runs in asynchronous mode. Returns -1 to indicate failure to initiate. Returns 0 to indicate success and then generates either the DCBEV_BRIDGEESTABLISHED termination event to indicate successful completion (conference bridge established), or the DCBEV_ERREVT in case of error. The DCBEV_BRIDGESTABLISHED event will have the TS_BRIDGECDT data structure as its associated data. The nBridgeID element in this structure can be used to identify the bridge. The DCBEV_ERREVT event will have the TS_BRIDGECDT data structure as its associated data. The BridgeID element in this structure can be used to identify the bridge ***Note:*** Use the Standard Runtime Library event management functions to handle the termination events. • EV_SYNC – Function runs in synchronous mode. Returns -1 on failure and 0 on success.
rfu	reserved for future use. Set this parameter to NULL.

Notes: ***1.*** Conference bridging can be used to effectively expand a conference beyond the maximum size allowed by your particular configuration. Before a conference reaches its maximum size, create a second (master) conference and then connect the two conferences via a bridge.

2. The total number of conference resources, as well as other conference resource limitations if any exist (such as the maximum conference size), depends upon either your particular board and its media load or the resource configuration of your HMP license. For specific conferencing resource information applicable to a particular board and its media load, see the *Configuration Guide* for DM3 architecture boards; or for HMP, see the release information (*Release Guide* or *Release Notes*).

3. Each conference bridge that is established consumes one conferencing resource within the master conference and one conferencing resource within the conference that is being bridged to the master conference.

4. It is possible to bridge together conferences that use the same DSP. In this case, the **hSrlDeviceA** and **hSrlDeviceB** parameters are the same.

5. The coach/pupil feature does not span conference bridges. Coach and pupil must be in the same conference.

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

Cautions

This function fails when:

- An invalid device handle is specified.

- Conference resources are not available on the DSP.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES 2

void main( )
{
   int dspdevh;              /*DSP device handle variable*/
   int tsdevhA1, tsdevhA2;   /*time slot device handles*/
   int tsdevhB1, tsdevhB2;   /*time slot device handles*/
   MS_CDT cdtA[NUM_PARTIES];   /*conference descriptor table for conference A*/
   MS_CDT cdtB[NUM_PARTIES];   /*conference descriptore table for conference B*/
   SC_TSINFO tsinfo;           /*time slot information data structure*/
   int nConferenceIDA;         /*Conference ID of conference A*/
   int nConferenceIDB;         /*Conference ID of conference B*/
   long scts;                  /*TDM bus time slot*/
   TS_BRIDGECDT Bridgecdt1;    /*Bridge CDT for bridge 1*/

   TS_BRIDGECDT objBridge;
   Bridgecdt1 = &objBridge;

   /*open conference board 1, DSP 2 device*/
   if ((dspdevh = dcb_open("dcbB1D2",0) == -1)) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /*open network board 1, time slot 1*/
   if ((tsdevhA1 = dt_open("dtiB1T1",0) == -1)) {
      printf("Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;
   if (dt_getxmitslot(tsdevhA1, &tsinfo) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevhA1));
      exit(1);
   }

   /*Set up CDT data structure*/
   cdtA[0].chan_num = (int)scts; /*scts is the time slot....*/
   cdtA[0].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtA[0].chan_attr = MSPA_TARIFF;
```

```
   /*open board 1, time slot 2*/
   if ((tsdevhA2 = dt_open("dtiB1T2",0) == -1)) {
      printf("Cannot open dtiB1T2: system error/n");
      exit(1);
   }

   if (dt_getxmitslot(tsdevhA2, &tsinfo) == -1)) {
      printf("dt_getxmitslot: Error Message = %s", ATDV_ERRMSGP(tsdevhA2));
      exit(1);
   }

   /*TDM bus time slot to be conferenced*/
   cdtA[1].chan_num = (int)scts; /*scts is the time slot....*/
   cdtA[1].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtA[1].chan_attr = MSPA_PUPIL; /*conferee may be coached later*/

   /*establish conference A*/
   if (dcb_estconf(dspdevh, cdtA, NUM_PARTIES, MSCA_ND, &nConferenceIDA) == -1) {
      printf("dcb_estconf: Error Message = %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /*open network board 1, time slot 1*/
   if ((tsdevhB1 = dt_open("dtiB1T1",0) == -1)) {
      printf("Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;
   }

   /*Set up CDT data structure*/
   cdtB[0].chan_num = (int)scts; /*scts is the time slot....*/
   cdtB[0].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtB[0].chan_attr = MSPA_TARIFF;

   /*open board 2, time slot 2*/
   if ((tsdevhB2 = dt_open("dtiB2T2",0) == -1)) {
      printf("Cannot open dtiB2T2: system error/n");
      exit(1);
   }

    /*TDM bus time slot to be conferenced*/
   cdtB[1].chan_num = (int)scts; /*scts is the time slot....*/
   cdtB[1].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtB[1].chan_attr = MSPA_PUPIL; /*conferee may be coached later*/

   /*establish conference B*/
   if (dcb_estconf(dspdevh, cdtB, NUM_PARTIES, MSCA_ND, &nConferenceIDB) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /*create bridge between conf A and conf B*/
   if (dcb_CreateBridge(dspdevh, nConferenceIDA, dspdevh, nConferenceIDB, Bridgecdt1, EV_ASYNC,
NULL)==-1)
   {
      printf("Error occurred during CreateBridge %s \n", ATDV_ERRMSGP(dspdevh));
   }
   else
   {
      printf("dcb_CreateBridge passed\n");
   }
```

```
   /*
    *Continue Processing
    */

   /*delete bridge between conf A and conf B*/
    if (dcb_DeleteBridge(dspdevh, nConferenceIDA, dspdevh, nConferenceIDB, Bridgecdt1, EV_ASYNC,
NULL)==-1)
   {
      printf("Error occurred during DeleteBridge %s \n", ATDV_ERRMSGP(dspdevh));
   }
   else
   {
      printf("dcb_DeleteBridge passed\n");
   }



   if (dt_close(tsdevhA1) == -1) {
      printf("Error closing tsdevh1 \n");
      exit(1);
   }

   if (dt_close(tsdevhA2) == -1) {
      printf("Error closing tsdevh2 \n");
      exit(1);
   }

   if (dt_close(tsdevhB1) == -1) {
      printf("Error closing tsdevh1 \n");
      exit(1);
   }

   if (dt_close(tsdevhB2) == -1) {
      printf("Error closing tsdevh2 \n");
      exit(1);
   }

   /*Delete the conference*/
   if (dcb_delconf(dspdevh, nConferenceIDA) == -1)) {
      printf("Cannot delete conference %d. Error message = %s", nConferenceIDA,
ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   if (dcb_delconf(dspdevh, nConferenceIDB) == -1)) {
      printf("Cannot delete conference %d. Error Message = %s", nConferenceIDB,
ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /*done processing--close device*/
   if (dcb_close(dspdevh) == -1) {
      printf("Cannot close DSP dcbB1D2: system error/n");
      exit(1);
   }

}
```

See Also

- dcb_DeleteBridge( )

dcb_delconf( )

Name: int dcb_delconf(devh, confid)

Inputs: int devh
- valid DSP device handle

int confid
- conference identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_delconf( )** function deletes a previously established conference. The conference identifier specifies the conference to be deleted.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	indicates the conference identifier of the conference to be deleted

Notes: ***1.*** Calling this function frees all resources in use by the conference.

2. Call the appropriate **xx_unlisten( )** function for each conferee before **dcb_delconf( )** is called.

Cautions

This function fails when:

- The specified device handle is invalid.
- The conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{
   int  dspdevh;                /* DSP device descriptor variable */
   int  tsdevh1, tsdevh2;       /* Time slot device descriptors */
   MS_CDT  cdt[NUM_PARTIES];    /* Conference descriptor table */
   int  confid;                 /* Conference ID */
   SC_TSINFO tsinfo;            /* Time slot information structure */
   long scts;                   /* TDM bus time slot */

   /* Open conference board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /* Assume DCB/SC connected to a DTI via TDM bus. */

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Get TDM bus transmit time slot of tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts ;      /* TDM bus time slot returned */
   cdt[0].chan_sel  = MSPN_TS;         /* from dt_getxmitslot() */
   cdt[0].chan_attr = MSPA_NULL;       /* Conferee has no special attributes */

   /* Open DTI board 1, time slot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf( "Cannot open dtiB1T2: system error/n");
      exit(1);
   }

   /* Get TDM bus transmit time slot of tsdevh2 */
   if (dt_getxmitslot(tsdevh2, &tsinfo) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /*  Set up CDT structure */
   cdt[1].chan_num  = (int)scts;       /* TDM bus time slot returned */
   cdt[1].chan_sel  = MSPN_TS;         /* from dt_getxmitslot() */
   cdt[1].chan_attr = MSPA_TARIFF;     /* Conferee receives periodic tariff tone */

   /* Establish a 2 party conference */
   if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }
```

```
   /* Do a listen for time slot tsdevh1 */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

   if (dt_listen(tsdevh1,&tsinfo) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Do a listen for the time slot tsdevh2 */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

   if (dt_listen(tsdevh2,&tsinfo) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Continue Processing */

   /* Unlisten the time slots */
   if (dt_unlisten(tsdevh1) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   if (dt_unlisten(tsdevh2) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Delete the conference */
   if (dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d. Error Message = %s", confid,
            ATDV_ERRMSGP(dspdevh));
      exit(1);
   }
   else
      printf("Conference deleted\n");

   /* Done processing - close all open devices */
   if(dcb_close(dspdevh) == -1) {
      printf("Cannot close dcbB1D2: system error/n");
      exit(1);
   }

   if(dt_close(tsdevh2) == -1) {
      printf("Cannot close dtiB1T2: system error/n");
      exit(1);
   }

   if(dt_close(tsdevh1) == -1) {
      printf("Cannot close dtiB1T1: system error/n");
      exit(1);
   }
}
```

See Also

- dcb_addtoconf( )
- dcb_estconf( )
- dcb_remfromconf( )

dcb_DeleteAllConferences( )

Name: int dcb_DeleteAllConferences(devh, mode, rfu)

Inputs:

int devh	• valid DSP device handle
unsigned short mode	• mode of the function
void* rfu	• void pointer that is reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous/asynchronous

Platform: DM3

Description

The **dcb_DeleteAllConferences( )** function deletes all established conferences. This function is provided for recovery purposes. A typical use would be if an application had to be restarted due to an abnormal termination: rather than downloading the firmware to re-initialize the boards, this function could be called to delete all conferences that might be left active in the firmware.

Parameter	Description
devh	indicates the valid device handle obtained when the DSP device was opened using **dcb_open( )**
mode	specifies whether the function should run asynchronously or synchronously. Possible values are as follows: • EV_ASYNC – Function runs in asynchronous mode. Returns -1 to indicate failure to initiate. Returns 0 to indicate successful initiation and then generates either the DCBEV_DELALLCONF termination event to indicate successful completion (all conferences deleted), or the DCBEV_ERREVT in case of error. ***Note:*** Use the Standard Runtime Library event management functions to handle the termination events. • EV_SYNC – Function runs in synchronous mode. Returns -1 on failure and 0 on success.
rfu	reserved for future use. Set this parameter to NULL.

Note: Calling this function frees all resources in use by all the conferences on the specified DSP device.

Cautions

- This function fails when the specified device handle is invalid.
- This function is intended for application program recovery purposes. (The **dcb_delconf( )** function is intended for standard conference management purposes and should be used with the appropriate **xx_unlisten( )** routing functions.)
- In a multi-threaded or multi-process environment, DCB functions should not be invoked at the same time **dcb_DeleteAllConferences( )** is called.
- When this function is executed, it will not perform **xx_unlisten( )** operations upon devices listening to conference time slots. It is the application's responsibility to perform unrouting using **xx_unlisten( )** functions.

Errors

if this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h.*

Example A

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>

#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

int dspdevh=0;

int  DeleteAllConferences(void);

void main(void)
{
  char DeviceName[16];
  sprintf(DeviceName,"dcbB1D1");
  printf("Trying to open device = %s\n",DeviceName);
  if ( (dspdevh = dcb_open(DeviceName, 0)) == -1) {
    printf("Cannot open device %s: system error/n", DeviceName);
    exit(1);
  }
  else  {
    printf("Open successfull for %s... dspdevh=0x%x\n",DeviceName,dspdevh);
  }

  /* Establish Conferences and Continue Processing  */

  /* Delete all Active Conferences  */
  DeleteAllConferences();

  /* Done processing - Close device */
  if ( dcb_close(dspdevh) == -1) {
    printf("Cannot close device %s: system error/n", DeviceName);
    exit(1);
  }
  else  {
```

```
    printf("dcb_close successfull ... dspdevh=0x%x\n",dspdevh);
  }
}  // main() ends


int DeleteAllConferences()
{
  void *RFU=0;
  if (dcb_DeleteAllConferences(dspdevh, EV_SYNC, RFU) == -1)
  {
    printf("dcb_DeleteAllConferences failed for %s.  Error message = %s",
            ATDV_NAMEP(dspdevh), ATDV_ERRMSGP(dspdevh));
    return(-1);
  }
  else
  {
    printf("dcb_DeleteAllConferences successfull for %s... \n",
            ATDV_NAMEP(dspdevh));
  }
  return(0);
}  //DeleteAllConferences ends
```

See Also

- **dcb_addtoconf( )**
- **dcb_delconf( )**
- **dcb_estconf( )**
- **dcb_remfromconf( )**

dcb_DeleteBridge( )

Name: int dcb_DeleteBridge(hSrlDeviceA, nConferenceIDA, hSrlDeviceB, nConferenceIDB, Bridgecdt, unMode, rfu)

Inputs:

int hSrlDeviceA	• valid DSP device handle of the master conference
int nConferenceIDA	• conference identifier for the master conference
int hSrlDeviceB	• valid DSP device handle of the conference that is bridged to the master conference
int nConferenceIDB	• conference identifier of the conference that is bridged to the master conference
TS_BRIDGECDT *Bridgecdt	• pointer to a conference bridge descriptor element
unsigned short unMode	• mode of the function
void *rfu	• void pointer that is reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: Asynchronous or synchronous

Platform: DM3

Description

The **dcb_DeleteBridge( )** function deletes a bridge that has been established between two conferences, but does not delete the individual conferences. The conferences that were connected via the conference bridge still exist after the bridge has been deleted, you must call the **dcb_delconf( )** to delete any remaining conferences.

Parameter	Description
hSrlDeviceA	specifies the valid device handle for the master conference obtained when the DSP device was opened using **dcb_open( )**
nConferenceIDA	indicates the conference identifier number of the master conference
hSrlDeviceB	specifies the valid DSP device handle used by a conference that is bridged to the master conference
nConferenceIDB	indicates the conference identifier number of a conference that is bridged to the master conference
Bridgecdt	points to the TS_BRIDGECDT data structure that defines the attributes of the conference bridge

Parameter	Description
unMode	specifies whether the function should run asynchronously or synchronously. Possible values are as follows: • EV_ASYNC – Function runs in asynchronous mode. Returns -1 to indicate failure to initiate. Returns 0 to indicate successful initiation and then generates either the DCBEV_BRIDGEREMOVED termination event to indicate successful completion (conference bridge deleted), or the DCBEV_ERREVT in case of error. The DCBEV_BRIDGESTABLISHED event will have the TS_BRIDGECDT data structure as its associated data. The BridgeID element in this structure can be used to identify the bridge. The DCBEV_ERREVT event will have the TS_BRIDGECDT data structure as its associated data. The BridgeID element in this structure can be used to identify the bridge ***Note:*** Use the Standard Runtime Library event management functions to handle the termination events. • EV_SYNC – Function runs in synchronous mode. Returns -1 on failure and 0 on success.
rfu	reserved for future use. Set this parameter to NULL.

Cautions

This function fails when:

- A specified device handle is invalid.
- A conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES 2

void main( )
{
   int dspdevh;              /*DSP device handle variable*/
   int tsdevhA1, tsdevhA2;   /*time slot device handles*/
   int tsdevhB1, tsdevhB2;   /*time slot device handles*/
   MS_CDT cdtA[NUM_PARTIES];   /*conference descriptor table for conference A*/
   MS_CDT cdtB[NUM_PARTIES];   /*conference descriptore table for conference B*/
```

```
SC_TSINFO tsinfo;           /*time slot information data structure*/
int nConferenceIDA;         /*Conference ID of conference A*/
int nConferenceIDB;         /*Conference ID of conference B*/
long scts;                  /*TDM bus time slot*/
TS_BRIDGECDT Bridgecdt1;    /*Bridge CDT for bridge 1*/

TS_BRIDGECDT objBridge;
Bridgecdt1 = &objBridge;

/*open conference board 1, DSP 2 device*/
if ((dspdevh = dcb_open("dcbB1D2",0) == -1)) {
   printf("Cannot open dcbB1D2: system error/n");
   exit(1);
}

/*open network board 1, time slot 1*/
if ((tsdevhA1 = dt_open("dtiB1T1",0) == -1)) {
   printf("Cannot open dtiB1T1: system error/n");
   exit(1);
}

tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;
if (dt_getxmitslot(tsdevhA1, &tsinfo) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevhA1));
   exit(1);
}

/*Set up CDT data structure*/
cdtA[0].chan_num = (int)scts; /*scts is the time slot....*/
cdtA[0].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
cdtA[0].chan_attr = MSPA_TARIFF;

/*open board 1, time slot 2*/
if ((tsdevhA2 = dt_open("dtiB1T2",0) == -1)) {
   printf("Cannot open dtiB1T2: system error/n");
   exit(1);
}

if (dt_getxmitslot(tsdevhA2, &tsinfo) == -1)) {
   printf("dt_getxmitslot: Error Message = %s", ATDV_ERRMSGP(tsdevhA2));
   exit(1);
}

/*TDM bus time slot to be conferenced*/
cdtA[1].chan_num = (int)scts; /*scts is the time slot....*/
cdtA[1].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
cdtA[1].chan_attr = MSPA_PUPIL; /*conferee may be coached later*/

/*establish conference A*/
if (dcb_estconf(dspdevh, cdtA, NUM_PARTIES, MSCA_ND, &nConferenceIDA) == -1) {
   printf("dcb_estconf: Error Message = %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/*open network board 1, time slot 1*/
if ((tsdevhB1 = dt_open("dtiB1T1",0) == -1)) {
   printf("Cannot open dtiB1T1: system error/n");
   exit(1);
}

tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;
}
```

```
   /*Set up CDT data structure*/
   cdtB[0].chan_num = (int)scts; /*scts is the time slot....*/
   cdtB[0].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtB[0].chan_attr = MSPA_TARIFF;

   /*open board 2, time slot 2*/
   if ((tsdevhB2 = dt_open("dtiB2T2",0) == -1)) {
      printf("Cannot open dtiB2T2: system error/n");
      exit(1);
   }

    /*TDM bus time slot to be conferenced*/
   cdtB[1].chan_num = (int)scts; /*scts is the time slot....*/
   cdtB[1].chan_sel = MSPN_TS; /*...returned from getxmitslot( )*/
   cdtB[1].chan_attr = MSPA_PUPIL; /*conferee may be coached later*/

   /*establish conference B*/
   if (dcb_estconf(dspdevh, cdtB, NUM_PARTIES, MSCA_ND, &nConferenceIDB) == -1) {
      printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /*create bridge between conf A and conf B*/
   if (dcb_CreateBridge(dspdevh, nConferenceIDA, dspdevh, nConferenceIDB, Bridgecdt1, EV_ASYNC,
NULL)==-1)
   {
      printf("Error occurred during CreateBridge %s \n", ATDV_ERRMSGP(dspdevh));
   }
   else
   {
      printf("dcb_CreateBridge passed\n");
   }

   /*
    *Continue Processing
    */

   /*delete bridge between conf A and conf B*/
    if (dcb_DeleteBridge(dspdevh, nConferenceIDA, dspdevh, nConferenceIDB, Bridgecdt1, EV_ASYNC,
NULL)==-1)
   {
      printf("Error occurred during DeleteBridge %s \n", ATDV_ERRMSGP(dspdevh));
   }
   else
   {
      printf("dcb_DeleteBridge passed\n");
   }



   if (dt_close(tsdevhA1) == -1) {
      printf("Error closing tsdevh1 \n");
      exit(1);
   }

   if (dt_close(tsdevhA2) == -1) {
      printf("Error closing tsdevh2 \n");
      exit(1);
   }

   if (dt_close(tsdevhB1) == -1) {
      printf("Error closing tsdevh1 \n");
      exit(1);
   }
```

```
   if (dt_close(tsdevhB2) == -1) {
      printf("Error closing tsdevh2 \n");
      exit(1);
   }

   /*Delete the conference*/
   if (dcb_delconf(dspdevh, nConferenceIDA) == -1)) {
      printf("Cannot delete conference %d. Error message = %s", nConferenceIDA,
ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   if (dcb_delconf(dspdevh, nConferenceIDB) == -1)) {
      printf("Cannot delete conference %d. Error Message = %s", nConferenceIDB,
ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /*done processing--close device*/
   if (dcb_close(dspdevh) == -1) {
      printf("Cannot close DSP dcbB1D2: system error/n");
      exit(1);
   }

}
```

See Also

- **dcb_CreateBridge( )**

dcb_dsprescount( )

Name: int dcb_dsprescount(devh, valuep)

Inputs:

int devh	• valid DSP device handle
int * valuep	• integer pointer to where the free DSP resource count is returned

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Configuration

Mode: synchronous

Platform: DM3

Description

The **dcb_dsprescount( )** function returns the available conferencing resource count for a specified DSP. The number of conferencing resources available depends on your conferencing board and the Media Load it is configured with.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
valuep	integer pointer to where the free DSP resource count is returned

Notes: ***1.*** A monitor is counted as one of the parties in a conference.

2. The total number of conference resources, as well as other conference resource limitations if any exist (such as the maximum conference size), depends upon either your particular board and its media load or the resource configuration of your HMP license. For specific conferencing resource information applicable to a particular board and its media load, see the *Configuration Guide* for DM3 architecture boards; or for HMP, see the release information (*Release Guide* or *Release Notes*).

3. Conference bridging can be used to effectively expand a conference beyond the maximum size allowed by your particular configuration. Before a conference reaches its maximum size, create a second (master) conference and then connect the two conferences via a bridge.

Calling any of the following functions will cause the available resource count to change:

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

dcb_CreateBridge( )
uses two resources for each bridge: one in the master conference and one in the conference that is bridged to the master conference

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

dcb_DeleteAllConferences( )
frees all resources in use by all the conferences on the specified DSP device

dcb_DeleteBridge( )
frees one resource in the master conference every time a conference bridge is deleted from the master conference

dcb_estconf( )
uses the number of resources as specified by the **numpty** parameter

dcb_monconf( )
uses one resource

dcb_remfromconf( )
frees one resource

dcb_unmonconf( )
frees one resource

Cautions

This function fails when the device handle specified is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

main()
{

   int  dspdevh;     /* DSP device descriptor */
   int  count;       /* DSP resource count */
```

```
   /* Open conference board 1, DSP 2 */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /* Get unused conference-resource count for dspdevh */
   if (dcb_dsprescount(dspdevh, &count) == -1) {
      printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }
   else
      printf("Free DSP Resource count = %d\n", count);

   if (dcb_close(dspdevh) == -1) {
      printf("Cannot close dcbB1D2: system error/n");
      exit(1);
   }
}
```

See Also

- dcb_addtoconf( )
- dcb_estconf( )
- dcb_delconf( )
- dcb_monconf( )
- dcb_remfromconf( )
- dcb_unmonconf( )

dcb_estconf( )

Name: int dcb_estconf(devh, cdt, numpty, confattr, confid)

Inputs:

int devh	• valid DSP device handle
MS_CDT *cdt	• pointer to conference descriptor table
int numpty	• initial number of parties in the conference
int confattr	• conference attributes
int *confid	• pointer to the conference identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
mslib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_estconf( )** function establishes a conference between parties. A conference is associated with a DSP and all resources used by the conference are on that DSP. When **dcb_estconf( )** returns successfully, **confid** will contain the conference identification number for use in all further modifications to that conference.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
cdt	points to the conference descriptor table. The conference descriptor table is an array of MS_CDT data structures which define the attributes of the conference parties.
numpty	indicates the initial number of parties in the conference. The maximum value is 24, but the maximum conference size may be lower depending upon your board and its media load or the resource configuration of your HMP license (see “Notes” that follow).

Parameter	Description
confattr	indicates a bitmask describing the properties of all parties in the conference. Valid values are as follows: • MSCA_ND – All parties in conference are notified by a tone if another conferee is added or removed from the conference. • MSCA_NULL – Conference has no special attributes. ***Note:*** The default MSCA_NULL must be used if the MSCA_ND conference attribute is not specified.
confid	points to the conference identifier number

Notes: ***1.*** Calling this function causes **numpty** resources to be used when the conference is successfully established.

2. This function may be used to initially establish a conference. You must use **dcb_addtoconf( )** or **dcb_monconf( )**to increase the size of the conference.

3. The maximum value for the **numpty** parameter is 24. If the maximum conference size is greater than 24 parties, you can use the **dcb_estconf( )** function to create a 24-party conference, and then use the **dcb_addtoconf( )** function to add parties one at a time.

4. If the maximum conference size is exceeded, the function generates an E_MSCNFLMT error (exceeds conference limit).

5. The total number of conference resources, as well as other conference resource limitations if any exist (such as the maximum conference size), depends upon either your particular board and its media load or the resource configuration of your HMP license. For specific conferencing resource information applicable to a particular board and its media load, see the *Configuration Guide* for DM3 architecture boards; or for HMP, see the release information (*Release Guide* or *Release Notes*).

6. Conference bridging can be used to effectively expand a conference beyond the maximum size allowed by your particular configuration. Before a conference reaches its maximum size, create a second (master) conference and then connect the two conferences via a bridge.

For each member of the conference, the TDM bus time slot number to listen to is returned in the chan_lts field of the MS_CDT data structure. The chan_attr field in the MS_CDT structure is redefined in the *msilib.h* file as follows:

```
#define   chan_lts     chan_attr
```

The chan_lts value must be used by the application to listen to the conferenced signal.

Cautions

This function fails when:

- An invalid device handle is specified.
- Conference resources are not available on the DSP.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

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

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   2

main()
{
   int  dspdevh;                        /* DSP Device handle variable */
   int  tsdevh1, tsdevh2;               /* Time slot device handles */
   MS_CDT  cdt[NUM_PARTIES];            /* Conference descriptor table */
   SC_TSINFO tsinfo;                    /* Time slot information structure */
   int  confid;                         /* Conference identifier */
   long scts;                           /* TDM bus time slot */

   /* Open conference board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2", 0) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /* Open network board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1", 0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts  ;    /* scts is the time slot... */
   cdt[0].chan_sel  = MSPN_TS    ;    /* ...returned from getxmitslot() */
   cdt[0].chan_attr = MSPA_TARIFF;
```

```
/* Open board 1, time slot 2 */
if ((tsdevh2 = dt_open("dtiB1T2", 0)) == -1) {
   printf( "Cannot open dtiB1T2: system error/n”);
   exit(1);
}

if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* SCbus time slot to be conferenced */
cdt[1].chan_num  = (int)scts ;     /* scts is the time slot... */
cdt[1].chan_sel  = MSPN_TS   ;     /* ...returned from getxmitslot() */
cdt[1].chan_attr = MSPA_PUPIL;     /* Conferee may be coached later */

/* Establish conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen() for the tsdevh1 to its conference signal */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen() for the tsdevh2 to its conference signal */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/*
 * Continue processing
 */

/* Unlisten the time slots */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

if (dt_close(tsdevh1) == -1){
   printf("Error closing tsdevh1\n");
   exit(1);
}
 if (dt_close(tsdevh2) == -1){
    printf("Error closing tsdevh2\n");
    exit(1);
}
```

```
   /* Delete the conference */
   if(dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d. Error Message = %s", confid,
               ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Done Processing - Close device */
   if(dcb_close(dspdevh) == -1) {
      printf("Cannot close DSP dcbB1D2: system error/n");
      exit(1);
   }
}
```

See Also

- **dcb_addtoconf( )**
- **dcb_delconf( )**
- **dcb_monconf( )**
- **dcb_remfromconf( )**
- **dcb_unmonconf( )**

dcb_evtstatus( )

Name: int dcb_evtstatus(event, action, status)

Inputs:

int event	• event identifier
int action	• action to be performed
int *status	• pointer to status of event generation

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Auxiliary

Mode: synchronous

Platform: DM3

Description

The **dcb_evtstatus( )** function gets or sets the status of a process-wide event. Certain features of the audio conferencing software are board-level features in that they are enabled or disabled on a per board basis. Process-wide events are enabled or disabled once for all devices used by that process.

Parameter	Description
event	indicates the specified process-wide event
action	specifies whether the event status is to be set or retrieved. Possible values are as follows: • SET_EVENT • GET_EVENT
status	If the event status is being set, ON or OFF is passed to the function in this parameter. If the event status is being retrieved, this parameter will contain ON or OFF when the function returns.

The **event** parameter must be set to MSG_RESTBL. MSG_RESTBL controls the Resource Table Update event generation. The resource assignment table is the mapping of resources to conferees. When this event notification is enabled, and the application makes a change to the assignment of resources on a conferencing board, a DCBEV_CTU event is generated. The updated resource table will be returned as the event data. Refer to the code example for details.

Cautions

dcb_evtstatus( ) is a process-wide function and does not have a device-handle as one of its parameters. Any event set ON or OFF is set for all devices used by the process, not for any particular device.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   MAX_PTY     32
#define   TABLE_SIZE 192

DCB_CT res_table[MAX_PTY];    /* DCB_CT structure array */

void handler()
{
   int   size = sr_getevtlen();
   char *datap = (char *)sr_getevtdatap();
   int   event_type = (int)sr_getevttype();

   printf("Event occurred on %s : Data size = %d : Data is at 0x%x\n",
           ATDV_NAMEP(sr_getevtdev()), size, datap);

   if (event_type == DCBEV_CTU) {
      memcpy(res_table, datap, TABLE_SIZE);
      }
   else {
      printf("unexpected event generated\n");
   }
}

main()
{
   int  bddevh, dspdevh;       /* conference board and DSP device descriptors */
   unsigned long atibits;      /* Active talker bits     */
   int  mode = SR_POLLMODE;    /* Standard Runtime Library function-call mode */
   unsigned int status;        /* conference board feature status */
   unsigned int i, count = 1000; /* Loop counters */

   /* Open conference board device */
   if ((bddevh = dcb_open("dcbB1",0)) == -1) {
      printf("Cannot open dcbB1: system error/n");
      exit(1);
   }
```

```
/* Set Standard Runtime Library function call mode */
if (sr_setparm(SRL_DEVICE, SR_MODEID, (void *)&mode) == -1){
    printf("Error setting sr_setparm()\n");
    exit(1);
}

/* Enable Standard Runtime Library event handler */
if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, (void *)handler) == -1) {
    printf("Error setting sr_enbhdlr()\n");
    exit(1);
}

/* Set Active Talker Identification On */
status = ACTID_ON;

if (dcb_setbrdparm(bddevh, MSG_ACTID, (void *)&status) == -1) {
   printf("Error setting board parameter - %s\n", ATDV_ERRMSGP(bddevh));
   exit(1);
}

/* Done with board-level calls : close device */
if (dcb_close(bddevh) == -1) {
   printf("Cannot close dcbB1: system error/n”);
   exit(1);
}

/* Set Resource Assignment Table Update events ON */
status = ON;

if (dcb_evtstatus(MSG_RESTBL, SET_EVENT, &status) == -1) {
   printf("Error enabling system-wide event\n");
   exit(1);
}

/* Open board 1, DSP 1 device */
if ((dspdevh = dcb_open("dcbB1D1",0)) == -1) {
   printf("Cannot open dcbB1D1: system error/n");
   exit(1);
}

/* Establish a conference and continue processing */

/* Wait in a 1000-count loop to get the active talkers */

while (count--) {
   if (dcb_getatibits(dspdevh, &atibits) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }
   printf("ATIBITS = %d\n", atibits);

   for (i=0; i<32; i++){
      if (atibits & (1<<i)){
         printf("confid = %d, TimeSlot = %d, Selector = %d\n",
                 res_table[i].confid, res_table[i].chan_num,
                 res_table[i].chan_sel);
      }
   } /* End of for() loop */
} /* End of while() loop */

/* Set Resource Table Update events OFF */
status = OFF;

if (dcb_evtstatus(MSG_RESTBL, SET_EVENT, &status) == -1) {
   printf("Error enabling system-wide event\n");
   exit(1);
}
```

```
    /* Disable event handler */
    if (sr_dishdlr(EV_ANYDEV, DCBEV_CTU, (void *)handler) == -1) {
        printf("Error in sr_dishdlr()\n");
        exit(1);
    }

    /* Done processing - close DSP device */

    if (dcb_close(dspdevh) == -1) {
       printf("Cannot close dcbB1D1: system error/n”);
       exit(1);
    }
}
```

See Also

- **dcb_gettalkers( )**

dcb_GetAtiBitsEx( )

Name: int dcb_GetAtiBitsEx(devh, numpty, ActiveTalkerInd, rfu)

Inputs:

int devh	• valid DSP device handle
int *numpty	• pointer to number of active talkers
DCB_CT *ActiveTalkerInd	• pointer to an array of active talker indicators
void *rfu	• reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Auxiliary

Mode: synchronous

Platform: DM3 (but not HMP)

■ Description

The **dcb_GetAtiBitsEx( )** function returns active talker indicators for all conferences on a DSP device at the time the function is called (i.e., who the active talkers are at a given moment). The current number of active talkers is returned in **numpty**, with information identifying the active talkers returned in **ActiveTalkerInd**.

Note: This function is not supported on Intel NetStructure® Host Media Processing (HMP) software.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
numpty	points to the number of active talkers indicators (elements in **ActiveTalkerInd**.
ActiveTalkerInd	points to an array of DCB_CT structures where active talker indicators are returned, which identify the active talkers

Notes: ***1.*** The developer must allocate and deallocate an array of DCB_CT data structures large enough to store information on the number of active talkers.

2. Active talker information is associated with the DSP device handle. The information is invalid upon closing the device.

■ Cautions

- This function is not supported on Intel NetStructure® Host Media Processing (HMP) software.

- The snapshot of information provided by **dcb_GetAtiBitsEx( )** is accurate for a split second. This information may not be accurate by the time the application processes it.
- This function fails when the device handle is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

const int MAX_ACTIVETALKERBITS = 120;

void main(void)
{
  char DeviceName[16];
  char BoardName[16];
  int dspdevh=-1,BoardDevHandle=-1;

  sprintf(BoardName,"dcbB1");
  printf("Trying to open device = %s\n",BoardName);
  if ( (BoardDevHandle = dcb_open(BoardName, 0)) == -1) {
    printf("Cannot open device %s: system error/n", BoardName);
    getchar();
    exit(1);
  }
  else {
    printf("Open successfull for %s ... BoardDevHandle=0x%x\n",BoardName,BoardDevHandle);
  }

  /* Set Active Talker Identification ON  */
  int nStatus=ACTID_ON;
  if(dcb_setbrdparm(BoardDevHandle,MSG_ACTID,(void*)&nStatus)==-1)  {
    printf("dcb_setbrdparm->MSG_ACTID->Error Setting Board Parm for %s : ERROR = %s\n",
          ATDV_NAMEP(BoardDevHandle),ATDV_ERRMSGP(BoardDevHandle));
    /* process error */
  }
  else  {
    printf("SetBoardParm->MSG_ACTID->Success Setting Board Parm for %s\n",
          ATDV_NAMEP(BoardDevHandle));
  }

  if ( dcb_close(BoardDevHandle) == -1) {
    printf("Cannot close device %s: system error/n", ATDV_NAMEP(BoardDevHandle));
    /* process error */
    exit(1);
  }
  else {
    printf("dcb_close successfull for %s... BoardDevHandle=0x%x\n",
          ATDV_NAMEP(BoardDevHandle),BoardDevHandle);
  }
```

```
  sprintf(DeviceName,"dcbB1D1");
  printf("Trying to open device = %s\n",DeviceName);
  if ( (dspdevh = dcb_open(DeviceName, 0)) == -1) {
    printf("Cannot open device %s: system error/n", DeviceName);
    /* process error  */
    exit(1);
  }
  else  {
    printf("Open successfull for %s... dspdevh=0x%x\n",DeviceName,dspdevh);
  }

  /* Establish Conferences and Continue Processing  */

  /*  GetAtiBitsEx  */
  int nCount,i=0;
  DCB_CT ActiveTalkerIndicators[MAX_ACTIVETALKERBITS];
  void * RFU=0;
  if(dcb_getatibitsEx(dspdevh, &nCount, ActiveTalkerIndicators, RFU)==-1)
  {
    printf("GetAtiBits->dcb_getatibitsEx failed on %s Error = %s\n",
          ATDV_NAMEP(dspdevh),ATDV_ERRMSGP(dspdevh));
    /* process error  */
  }
  else
  {
    printf("GetAtiBits->dcb_getatibitsEx Successful on %s Count = %d\n",
          ATDV_NAMEP(dspdevh),nCount);
    for(i=0;i<nCount;i++)
    {
      printf("i = %d ConferenceID = 0x%x ChanNum = %d ChanSel = 0x%x\n",
            i,ActiveTalkerIndicators[i].confid,
            ActiveTalkerIndicators[i].chan_num,
            ActiveTalkerIndicators[i].chan_sel);
    }
  }

  /* Done processing - Close device */
  if ( dcb_close(dspdevh) == -1) {
    printf("Cannot close device %s: system error/n", ATDV_NAMEP(dspdevh));
    /* process error  */
    exit(1);
  }
  else
    printf("dcb_close successfull for %s... dspdevh=0x%x\n",
          ATDV_NAMEP(dspdevh),dspdevh);
}  // main() ends
```

See Also

- **dcb_evtstatus( )**
- **dcb_gettalkers( )**
- **dcb_dsprescount( )**

dcb_getbrdparm( )

Name: int dcb_getbrdparm(devh, param, valuep)

Inputs:

int devh	• valid board device handle
unsigned char param	• device parameter defined name
void * valuep	• pointer to the returned parameter value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Configuration

Mode: synchronous

Platform: DM3

Description

The **dcb_getbrdparm( )** function retrieves a conference board parameter value. Each parameter has a symbolic name that is defined in *dcblib.h*. The parameters are disabled by default and must be enabled using the **dcb_setbrdparm( )** function.

Parameter	Description
devh	specifies the valid device handle obtained when the board device was opened using **dcb_open( )**
param	indicates the parameter to be examined
valuep	points to the integer or MS_VOL data structure where the value of the parameter specified in **param** should be returned

The valid values for **param** and **valuep** are shown below:

Note: For MSG_ACTID, MSG_ACTTALKERNOTIFYINTERVAL, and MSG_TONECLAMP, **valuep** points to an integer value. For MSG_VOLDIG, **valuep** points to an MS_VOL data structure.

MSG_ACTID (Active Talker Identification)

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

conference. Active talkers are determined by their loudness; i.e., the strength of their "non-silence" energy.

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

MSG_ACTTALKERNOTIFYINTERVAL (Active Talker Notification Interval)

Changes the interval specifying how frequently the Active Talker status is updated. The value is specified in 10 ms units. The default value is 100 (in 10 ms units), which results in a 1-second interval, and the maximum value is 1000, which results in a 10-second interval.

Note: If a low value is used, it can affect system performance due to the more frequent updating of the status (which results in a high quantity of internal notification messages). If a high value is used, it will result in less frequent updating of status, but the non-silence energy of a conferee may not be reported if it occurs between notification updates. For example, if the notification interval is set to 2 seconds and a conferee only says "yes" or "no" quickly in between notifications, that vocalization by the conferee will not be reported.

MSG_TONECLAMP (Tone Clamp Activation)

Enables tone clamping for all parties to reduce the amount of DTMF tones heard in a conference. Tone clamping applies to the transmitted audio going into the conference and does not affect DTMF function. It is meaningful only in the full duplex or the transmit-only mode. Possible values are TONECLAMP_ON or TONECLAMP_OFF. TONECLAMP_OFF is the default. (To enable on a per-party basis, set the MSPA_PARTY_TONECLAMP attribute in the MS_CDT structure for the party.) Even with tone clamping, 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.

MSG_VOLDIG (Volume Control Digits)

Defines the volume control status and volume up/down/reset digits as defined in the MS_VOL data structure.

Cautions

- The value of the parameter returned by this function is currently an integer or an MS_VOL data structure. **valuep** is the address of the value, but should be cast as a void pointer when passed in the value field.
- This function fails when:
  - The device handle is invalid.
  - The parameter specified is invalid.
  - The DSP device handle is used.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

.Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

main()
{
   int  brddevh;      /* Board dev descriptor variables */
   int  actid_status; /* Active talker identification status (ON/OFF)  */
   MS_VOL  volume;    /* Volume control structure */

   /* Open DCB board 1 */
   if ((brddevh = dcb_open("dcbB1",0)) == -1) {
      printf( "Cannot open dcbB1: system error/n");
      exit(1);
   }

   /* Retrieve Status (ON/OFF) of the Active Talker Identification */
   if (dcb_getbrdparm(brddevh, MSG_ACTID, (void *)&actid_status) == -1) {
      printf("Error getting board param:0x%x\n ", ATDV_LASTERR(brddevh));
      exit(1);
   }
   printf("Active talker identification is %s\n", (actid_status ? "ON" : "OFF"));

   /* Retrieve Information on Volume Control Feature */
   if (dcb_getbrdparm(brddevh, MSG_VOLDIG,(void *)&volume) == -1) {
      printf("Error getting volume control parameters : 0x%x\n ",
             ATDV_LASTERR(brddevh));
      exit(1);
   }
   printf("Volume Control is %s\n", (volume.vol_control ? "ON" : "OFF"));
   printf("The Up Digit is       %d\n", volume.vol_up);
   printf("The Reset Digit is    %d\n", volume.vol_reset);
   printf("And the Down Digit is %d\n", volume.vol_down);

   /* Continue processing */

   if (dcb_close(brddevh)== -1){
      printf("Cannot close dcbB1: system error/n");
      exit(1);
   }
}
```

See Also

- **dcb_setbrdparm( )**
- **dcb_GetPartyParm( )**
- **dcb_SetPartyParm( )**

dcb_getcde( )

Name: int dcb_getcde(devh, confid, cdt)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
MS_CDT *cdt	• pointer to a conference descriptor table element

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_getcde( )** function retrieves the properties of a conferee in an existing conference.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	indicates the conference identifier number
cdt	points to an MS_CDT data structure that specifies the attributes of the conferee

This function requires that the conferee's chan_num and chan_sel be specified in the MS_CDT data structure. On successful completion, the conferee's attributes will be returned in the chan_attr field.

Note: This function must be invoked multiple times if the attributes of more than one conferee are desired.

Cautions

This function fails when:

- The device handle specified is invalid.
- An invalid conference identifier is specified.
- The queried conferee is not in the conference.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{
   int    dspdevh;            /* DSP device handle */
   int    tsdevh1, tsdevh2;   /* DTI time slot device handles */
   MS_CDT cdt[NUM_PARTIES];   /* Conference descriptor table */
   int    confid;             /* Conference identifier */
   int    attrib;             /* Time slot attribute */
   long   scts1, scts2;       /* TDM bus time slots */

   /* Open conference board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Prepare the time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts1;

   /* Retrieve the TDM bus transmit time slot for tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up the CDT structure */
   cdt[0].chan_num  = (int)scts1;        /* scts is the TDM bus transmit time slot */
   cdt[0].chan_sel  = MSPN_TS;          /* returned from dt_getxmitslot(). */
   cdt[0].chan_attr = MSPA_TARIFF;      /* Conferee will receive periodic tariff tone */

   /* Open DTI board 1, tslot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf( "Cannot open dtiB1T2: system error/n");
      exit(1);
   }
```

```
/* Prepare the time slot information structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts2;

if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* TDM bus time slot to be conferenced */
cdt[1].chan_num  = (int)scts2;      /* scts is the TDM bus transmit time slot */
cdt[1].chan_sel  = MSPN_TS;        /* returned from dt_getxmitslot(). */
cdt[1].chan_attr = MSPA_PUPIL;     /* The conferee may be coached later */

/* Establish the two party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen to the conference signal for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen to the conference signal for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Now get the attribute of conferee on tsdevh2 */
cdt[0].chan_num = (int)scts2;
cdt[0].chan_sel = MSPN_TS;

if(dcb_getcde(dspdevh, confid, &cdt)==-1){
  printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
  exit(1);
}
printf ("%s has conferee attribute 0x%x\n", ATDV_NAMEP(tsdevh2), cdt[0].chan_attr);

/* Finished with conference, so remove listens */

if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Delete the conference */
if(dcb_delconf(dspdevh, confid) == -1) {
   printf("Cannot delete conference %d. Error Message = %s", confid,
           ATDV_ERRMSGP(dspdevh));
   exit(1);
}
```

```
   /* And close all open devices */
   if (dt_close(tsdevh1) == -1){
      printf("Cannot close tsdevh1: system error/n”);
      exit(1);
   }

   if (dt_close(tsdevh2) == -1){
      printf("Cannot close tsdevh2: system error/n”);
      exit(1);
   }

   if (dcb_close(dspdevh) == -1){
      printf("Cannot close dcbB1D2: system error/n”);
      exit(1);
   }
}
```

See Also

- **dcb_setcde( )**

dcb_getcnflist( )

Name: int dcb_getcnflist(devh, confid, numpty, cdt)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
int *numpty	• pointer to the conferee count
MS_CDT *cdt	• pointer to conference descriptor table

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_getcnflist( )** function retrieves total number of parties within a conference and a conferee list. The list contains specific information about each conferee in that conference, including each conferee's TDM bus transmit time slot number, selector, and conferee attribute description. The list is not returned in any specific order.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference identifier
numpty	points to the conferee count
cdt	points to the conference descriptor table. The conference descriptor table is an array of MS_CDT data structures, which specify the attributes of the conference parties.

Note: The application is responsible for allocating an MS_CDT table with sufficient elements.

When a conference is being monitored, one member of the conference list will be the monitor. chan_num for the monitor will equal 0x7FFF and chan_sel will be MSPN_TS.

If you call this function to get the number of conferees in a conference that contains a conference bridge, the return value will be the total number of conferees in the conference plus one for the conference bridge, or in the case of a master conference, one for each bridge that is connected to the master conference.

Cautions

- This function fails when an invalid conference identifier is specified.
- It is the responsibility of the application to allocate enough memory for the conference descriptor table. There must be an MS_CDT element allocated for each conferee description returned by this function. For example, if a conference was started with four conferees, and three conferees were added later, the MS_CDT array must be able to hold seven entries.

Note: Even though **dcb_monconf( )** does not use the CDT structure, the array must have an additional structure if the conference is being monitored.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{

   int  dspdevh;            /* DSP device handle variable */
   int  tsdevh1, tsdevh2;   /* DTI TDM bus time slot device handles */
   int  partycnt;           /* Pointer to the number of conferenced parties */
   MS_CDT  cdt[32];         /* Conference descriptor table */
   SC_TSINFO tsinfo;
   int  confid;             /* Conference identifier */
   long  scts;              /* Returned TDM bus time slot */
   int  i;                  /* Loop index */

   /* Open board 1 DSP 1 device */
   if ((dspdevh = dcb_open("dcbB1D1",0)) == -1) {
      printf( "Cannot open dcbB1D1: system error/n");
      exit(1);
   }

   /* Assume the conference board connected to a DTI via TDM bus. */

   /* Open board 1, tslot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
       printf( "Cannot open dtiB1T1: system error/n");
       exit(1);
   }

   /* Prepare the TDM bus time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;
```

```
/* Retrieve the TDM bus transmit time slot for tsdevh1 */
if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Set up the CDT structure */
cdt[0].chan_num  = (int)scts;        /* scts is the TDM bus time slot */
cdt[0].chan_sel  = MSPN_TS;          /* returned from dt_getxmitslot() */
cdt[0].chan_attr = MSPA_NULL;        /* Conferee has no special attributes */

/* Open board 1, tslot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf( "Cannot open dtiB1T2: system error/n");
   exit(1);
}

/* Retrieve the TDM bus transmit time slot for tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message = %s" , ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* TDM bus time slot to be conferenced */
cdt[1].chan_num  = (int)scts;        /* scts is the SCbus time slot */
cdt[1].chan_sel  = MSPN_TS;          /* returned from dt_getxmitslot() */
cdt[1].chan_attr = MSPA_PUPIL;       /* Conferee may be coached later */

/* Establish 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen to the TDM bus listen time slot for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen to the TDM bus listen time slot for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Get conferee list */
if (dcb_getcnflist(dspdevh, confid, &partycnt, &cdt[0]) == -1) {
   printf("Error Message = %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Display conference information */
printf("Number of parties in conference %d = %d\n", confid, partycnt);

for (i=0; i<partycnt; i++){
    printf("%d : Chan_num = 0x%x", i+1, cdt[i].chan_num);
    printf("     Chan_sel = 0x%x",      cdt[i].chan_sel);
    printf("     Chan_att = 0x%x\n",    cdt[i].chan_attr);
}
```

```
   /* Remove all listens */
   if (dt_unlisten(tsdevh1) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   if (dt_unlisten(tsdevh2) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Delete the conference */
   if(dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d : Error Message = %s", confid, ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Close all open devices */
   if (dt_close(tsdevh1) == -1){
      printf("Cannot close tsdevh1: system error/n”);
      exit(1);
   }

   if (dt_close(tsdevh2) == -1){
      printf("Cannot close tsdevh2: system error/n”);
      exit(1);
   }

   if (dcb_close(dspdevh)== -1){
      printf("Cannot close dcbB1D1: system error/n”);
      exit(1);
   }
}
```

See Also

- **dcb_estconf( )**

dcb_getdigitmsk( )

Name: int dcb_getdigitmsk(devh, confid, bitmaskp)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
unsigned int * bitmaskp	• pointer to digit bitmask

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Configuration

Mode: synchronous

Platform: DM3

Description

The **dcb_getdigitmsk( )** function returns the digit mask for a specified conference. The values set in the mask corresponds to the digits which, when received, will cause a DCBEV_DIGIT event to be generated to the application.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	indicates the conference identifier number
bitmask	points to the digit bitmask

Note: If MSG_VOLDIG is enabled to give transparent volume control to the conferees, the digits for volume increase, decrease, and reset will not cause digit events to be generated. As a result, the application will not know if the volume changes.

Cautions

This function fails when:

- The device handle specified is invalid
- An invalid conference identifier is specified

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   2

main()
{
   int  dspdevh;              /* DSP device handle */
   int  confid;               /* Conference Identifier */
   unsigned int bitmask;      /* bitmask variable */
   int  tsdevh1, tsdevh2;     /* DTI time slot device handles */
   MS_CDT cdt[NUM_PARTIES];   /* Conference descriptor table */
   SC_TSINFO tsinfo;
   long scts;                 /* TDM bus time slot */

   /* Open confernce board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Prepare the time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* Retrieve the TDM bus transmit time slot for tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up the CDT[0] structure */
   cdt[0].chan_num  = (int)scts;   /* scts is the TDM bus time slot */
   cdt[0].chan_sel  = MSPN_TS;     /* returned from dt_getxmitslot() */
   cdt[0].chan_attr = MSPA_TARIFF; /* Party receives periodic tariff tone */

   /* Open DTI board 1, tslot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf( "Cannot open dtiB1T2: system error/n");
      exit(1);
   }
```

```
/* Retrieve the TDM bus transmit time slot for tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up the CDT[1] structure */
cdt[1].chan_num  = (int)scts;           /* scts is the TDM bus time slot */
cdt[1].chan_sel  = MSPN_TS;             /* returned from dt_getxmitslot() */
cdt[1].chan_attr = MSPA_PUPIL;          /* Conferee may be coached later */

/* Establish a two party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for the tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for the tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Enable digit detection for digits 1,3 and 5 only */

   if (dcb_setdigitmsk(dspdevh, confid, CBMM_ONE | CBMM_THREE | CBMM_FIVE,
       CBA_SETMSK)) == -1) {
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Get the bitmask value for the digit detection event */
if (dcb_getdigitmsk(dspdevh, confid, &bitmask) == -1) {
    printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
    exit(1);
}
 /*
 *   Display list of digits enabled for detection
 */
if (bitmask & CBMM_ZERO)
   printf("Digit 0 is enabled\n");
if (bitmask & CBMM_ONE)
   printf("Digit 1 is enabled\n");
if (bitmask & CBMM_TWO)
   printf("Digit 2 is enabled\n");
if (bitmask & CBMM_THREE)
   printf("Digit 3 is enabled\n");
if (bitmask & CBMM_FOUR)
   printf("Digit 4 is enabled\n");
if (bitmask & CBMM_FIVE)
   printf("Digit 5 is enabled\n");
if (bitmask & CBMM_SIX)
   printf("Digit 6 is enabled\n");
if (bitmask & CBMM_SEVEN)
   printf("Digit 7 is enabled\n");
```

```
   if (bitmask & CBMM_EIGHT)
      printf("Digit 8 is enabled\n");
   if (bitmask & CBMM_NINE)
      printf("Digit 9 is enabled\n");
   if (bitmask & CBMM_STAR)
      printf("Digit * is enabled\n");
   if (bitmask & CBMM_POUND)
      printf("Digit # is enabled\n");
   if (bitmask & CBMM_A)
      printf("Digit A is enabled\n");
   if (bitmask & CBMM_B)
      printf("Digit B is enabled\n");
   if (bitmask & CBMM_C)
      printf("Digit C is enabled\n");
   if (bitmask & CBMM_D)
      printf("Digit D is enabled\n");

   /* Unlisten the time slots */
   if (dt_unlisten(tsdevh1) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   if (dt_unlisten(tsdevh2) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Delete the conference */
   if(dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d. Error Message = %s", confid,
               ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Done Processing - Close all open devices */

   if (dt_close(tsdevh1) == -1){
      printf("Error closing tsdevh1\n");
      exit(1);
   }

   if (dt_close(tsdevh2) == -1){
      printf("Error closing tsdevh2\n");
      exit(1);
   }

   if(dcb_close(dspdevh) == -1) {
     printf("Cannot close dcbB1D2: system error/n”);
     exit(1);
   }
}
```

See Also

- .dcb_setdigitmsk( )
- dcb_setbrdparm( )

dcb_GetPartyParm( )

Name: int dcb_GetPartyParm(hSrlDevice, nConferenceID, Partycdt, unPartyParm, Value, rfu)

Inputs:

int hSrlDevice	• valid DSP device handle
int nConferenceID	• conference identifier
MS_CDT *Partycdt	• pointer to conference descriptor table element
unsigned int unPartyParm	• the parameter whose value is to be retrieved
void *Value	• the address of the integer or structure containing the value(s) to be assigned
void *rfu	• void pointer reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3 (but not HMP)

Description

The **dcb_GetPartyParm( )** function retrieves the current parameters for a conferee (conference party).

Note: This function is **not** supported on Intel NetStructure® Host Media Processing (HMP) software.

Parameter	Description
hSrlDevice	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
nConferenceID	specifies the conference identifier number
Partycdt	points to an MS_CDT data structure that defines the attributes of the conferee *Note:* The chan_attr field of the MS_CDT structure is not used and is ignored by the **dcb_SetPartyParm( )** function.
unPartyParm	specifies the parameter whose value is to be retrieved
Value	points to the integer or structure containing the value to be assigned to **unPartyParm**
rfu	void pointer reserved for future use

For a list of valid values for **unPartyParm** and **Value**, see the **dcb_SetPartyParm( )** function description.

Cautions

- This function is **not** supported on Intel NetStructure® Host Media Processing (HMP) software.
- This function fails when:
  - The device handle specified is invalid.
  - The conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"
#define   NUM_PARTIES   2

main()
{
   int  dspdevh = -1;            /* DSP device handle */
   MS_CDT cdt[NUM_PARTIES]={0};  /* Conference descriptor table */
   int  confid = -1;             /* Conference ID */
   int  tsdevh1=-1, tsdevh2=-1;  /* DTI time slot device handle */
   long  scts = -1;              /* TDM bus transmit time slot */
   SC_TSINFO  tsinfo = {0};      /* Time slot information structure */
   int vol;                      /* Conferee volume level */

   /* Open conference board 1, DSP 3 device */
   if ((dspdevh = dcb_open("dcbB1D3",0)) == -1) {
      printf("Cannot open dcbB1D3 : system error/n");
      exit(1);
   }
   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }
   /* Prepare time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;
   /* get transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }
   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts;     /* TDM bus time slot returned */
   cdt[0].chan_sel  = MSPN_TS;       /* by dt_getxmitslot() */
```

```
/* Open DTI board 1, time slot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf( "Cannot open dtiB1T2 : system error/n");
   exit(1);
}
/* Get transmit time slot of DTI tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Set up CDT structure */
cdt[1].chan_num  = (int)scts;    /* TDM bus time slot returned */
cdt[1].chan_sel  = MSPN_TS;      /* returned from getxmitslot */
/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
   exit(1);
}
/* Do a listen for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;
if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;
if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Now get the volume level of the 1st conferee */
/* NOTE: scts still contains the transmit time slot of tsdevh2 */

cdt[0].chan_num = (int)scts;
cdt[0].chan_sel = MSPN_TS;
if((dcb_GetPartyParm(dspdevh, confid, &cdt[0], MSPA_OUTPUTVOLABS, &vol, 0)) == -1) {
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}
/* Print out the current volume */
printf ("The volume is currently set to %d\n", vol);

/* Perform 'unlistens' on the listening DTI time slots */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}
if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Delete the conference */
if(dcb_delconf(dspdevh, confid) == -1) {
   printf("Cannot delete conference %d. Error Message = %s", confid, ATDV_ERRMSGP(dspdevh));
   exit(1);
}
/* Close all open devices */
if (dt_close(tsdevh1) == -1){
   printf("Error closing tsdevh1\n");
   exit(1);
}
if (dt_close(tsdevh2) == -1){
   printf("Error closing tsdevh2\n");
   exit(1);
```

```
    }
    if (dcb_close(dspdevh) == -1){
       printf("Cannot close dcbB1D3 : system error/n");
       exit(1);
    }

}
```

See Also

- dcb_SetPartyParm( )

dcb_gettalkers( )

Name: int dcb_gettalkers(devh, confid, numpty, talkers)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
int * numpty	• pointer to number of active talkers
MS_CDT * talkers	• pointer to array of talker descriptions (party attributes)

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Auxiliary

Mode: synchronous

Platform: DM3

Description

The **dcb_gettalkers( )** function retrieves information about the conferees actively talking in the conference specified by the DSP device and the **confid**. The current number of active talkers is returned in **numpty**, with descriptive information on each talker returned in **talkers**. The returned array of MS_CDT structures contains the active talker party attributes. The array has **numpty** number of elements. Each MS_CDT structure describes one active talker. chan_num contains the transmit time slot number of the actively talking conferee. chan_sel specifies that the conferee is a TDM bus time slot. For active talker retrieval, chan_attr is not used.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	indicates the conference identifier number
numpty	points to number of active talkers
talkers	points to the array of MS_CDT data structures that contain active talker descriptions

Notes: ***1.*** Active talker information is associated with the DSP device handle. The information is invalid upon closing the device.

2. The developer must allocate and deallocate an array of MS_CDT data structures large enough to store information on the number of active talkers.

3. The list is not returned in any specific order.

Cautions

This function fails when:

- The device handle specified is invalid.
- An invalid conference identifier is specified.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   2
#define   MAX_PTY       32

main()
{
   int  dspdevh;           /* DSP device handle */
   int  tsdevh1;           /* DTI time slot device handle */
   int  partycnt;          /* The no. of conferenced parties */
   int  confid;            /* Conference identifier */
   SC_TSINFO     tsinfo;   /* Time slot information structure */
   MS_CDT  cdt[MAX_PTY];   /* Conference descriptor table */
   long  scts;             /* TDM bus time slot */
   int   i;                /* Loop index */

   /* Open conference board 1, DSP 1 device */
   if ((dspdevh = dcb_open("dcbB1D1",0)) == -1) {
      printf("Cannot open dcbB1D1: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 device */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf("Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 2 device */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf("Cannot open dtiB1T2: system error/n");
      exit(1);
   }

   /* Prepare time slot information structure */
   tsinfo.sc_numts=1
   tsinfo.sc_tsarrayp=&scts;
```

```
/* Get conference transmit time slot of tsdevh1 */
if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Set up CDT structure, for tsdevh1 */
cdt[0].chan_num = (int)scts;    /* TDM bus time slot returned */
cdt[0].chan_sel = MSPN_TS;      /* ...by dt_getxmitslot() */
cdt[0].chan_attr = MSPA_NULL;   /* Conferee has no special attributes */

/* Get TDM bus transmit time slot of tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up CDT structure, for tsdevh2 */
cdt[1].chan_num = (int)scts;    /* TDM bus time slot returned */
cdt[1].chan_sel = MSPN_TS;      /* by dt_getxmitslot() */
cdt[1].chan_attr = MSPA_PUPIL;  /* Conferee may be coached later */

/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1) {
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Prepare time slot information structure */
tsinfo.sc_numts=1
tsinfo.sc_tsarrayp=cdt[0].chan_lts;

/* Listen to the time slot returned by dcb_estconf() */
if (dt_listen(tsdevh1, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Prepare time slot information structure */
tsinfo.sc_numts=1
tsinfo.sc_tsarrayp=cdt[1].chan_lts;

/* Listen to the time slot returned by dcb_estconf() */
if (dt_listen(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Find out who is currently talking */
if ((dcb_gettalkers(dspdevh,confid,&partycnt,&cdt)) == -1) {
   printf ("Error Message = %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Print out the time slot numbers of currently active talkers */
printf ("There are %d currently active talkers\n", partycnt);
for (i=0; i<partycnt; i++){
   printf ("Time slot = %d , Chan_sel = 0x%x\n", cdt[i].chan_num, cdt[i].chan_sel);
}

/* Remove all time slot listens */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}
```

```
   if (dt_unlisten(tsdevh2) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Delete the conference */
   if(dcb_delconf(dspdevh, confid) == -1) {
      printf("Cannot delete conference %d. Error Message = %s", confid,
              ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Done processing - close all open devices */
   if (dt_close(tsdevh1) == -1) {
      printf("System error closing %s/n”, ATDV_NAMEP(tsdevh1));
      exit(1);
   }

   /* Done processing - close device */
   if (dt_close(tsdevh2) == -1) {
      printf("System error closing %s/n", ATDV_NAMEP(tsdevh2));
      exit(1);
   }

   /* Done processing - close device */
   if (dcb_close(dspdevh) == -1) {
      printf("Cannot close dcbB1D1: system error/n");
      exit(1);
   }
}
```

See Also

- **dcb_GetAtiBitsEx( )**
- **dcb_dsprescount( )**

dcb_monconf( )

Name: int dcb_monconf(devh, confid, lts)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
long *lts	• pointer to listen TDM bus time slot

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_monconf**( ) function adds a monitor to a conference. A monitor has no input in the conference.

This function places the monitored signal on the TDM bus. Several parties can listen to the monitored signal simultaneously.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference identifier number
lts	points to the returned listen TDM bus time slot. The monitored signal is present on this time slot.

Notes: ***1.*** There may only be one monitor in a conference. The monitor feature does not span conference bridges.

2. Calling this function uses one conferencing resource.

3. It is the application's responsibility to listen to the time slot on which the monitored signal is transmitted.

A monitor counts as one of the parties in the conference. If all the resources on the DSP are already in use, it is not possible to monitor the conference. When a conference is deleted, the conference monitor is also deleted.

Cautions

This function fails when:

- The device handle specified is invalid.
- The conference is full.
- Conference resources are not available on the DSP.
- The conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{
   int  dspdevh;               /* DSP device handle */
   int  tsdevh1, tsdevh2;      /* DTI time slot device handles */
   MS_CDT  cdt[NUM_PARTIES];   /* Conference descriptor table */
   int  confid;                /* Conference identifier */
   long  lts, scts;            /* TDM bus listen/transmit time slots */
   SC_TSINFO  tsinfo;          /* Time slot information structure */

   /* Open conference board 1, DSP 3 device */
   if ((dspdevh = dcb_open("dcbB1D3",0) == -1) {
      printf("Cannot open dcbB1D3: system error/n");
      exit(1);
   }

   /* Open DTI board 1, tslot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
         printf( "Cannot open dtiB1T1: system error/n");
         exit(1);
   }

   /* Prepare the TDM bus time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* Get TDM bus transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message = %s",ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }
```

```
/* Set up CDT structure */
cdt[0].chan_num  = (int)scts;   /* SCbus transmit time slot returned */
cdt[0].chan_sel  = MSPN_TS;     /* ...from dt_getxmitslot() */
cdt[0].chan_attr = MSPA_NULL;   /* Conferee has no special attributes */

/* Open DTI board 1, tslot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf("Cannot open dtiB1T2: system error/n");
   exit(1);
}

/* Get transmit time slot of DTI tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up CDT structure */
cdt[1].chan_num  = (int)scts ;   /* SCbus time slot returned */
cdt[1].chan_sel  = MSPN_TS;      /* from dt_getxmitslot() */
cdt[1].chan_attr = MSPA_NULL;    /* Conferee has no special attributes */

/* Open DTI board 1, tslot 3 */
if ((tsdevh1 = dt_open("dtiB1T3",0)) == -1) {
     printf( "Cannot open dtiB1T3: system error/n");
     exit(1);
}

/* Establish 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for the tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for the tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Now monitor the conference on time slot lts */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &lts;

if((dcb_monconf(dspdevh,confid,&lts)) == -1){
  printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
  exit(1);
}

/* Assume that a DTI time slot, tsdevh3, is a monitor */
if (dt_listen(tsdevh3,&tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh3));
   exit(1);
}
```

```
    /* Perform an unlisten() to end monitor listening */
    if (dt_unlisten(tsdevh3) == -1){
        printf("Error Message : %s", ATDV_ERRMSGP(tsdevh3));
        exit(1);
    }

    /* Now remove the monitor from the conference */
    if((dcb_unmonconf(dspdevh,confid)) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
    }

    /* 'Unlisten' the TDM bus time slots */
    if (dt_unlisten(tsdevh1) == -1){
       printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
       exit(1);
    }

    if (dt_unlisten(tsdevh2) == -1){
       printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
       exit(1);
    }

    /* Delete the conference */
    if(dcb_delconf(dspdevh, confid) == -1) {
       printf("Cannot delete conference %d : Error Message = %s", confid,
       ATDV_ERRMSGP(dspdevh));
       exit(1);
    }

    /* Close all open devices */
    if (dt_close(tsdevh1) == -1){
       printf("Error closing tsdevh1\n");
       exit(1);
    }

    if (dt_close(tsdevh2) == -1){
       printf("Error closing tsdevh2\n");
       exit(1);
    }

    if (dt_close(tsdevh3) == -1){
       printf("Error closing tsdevh3\n");
       exit(1);
    }

    if (dcb_close(dspdevh) == -1){
       printf("Cannot close dcbB1D3: system error/n”);
       exit(1);
    }
}
```

See Also

- **dcb_unmonconf( )**

dcb_open( )

Name: int dcb_open(name, rfu)

Inputs: char *name
- pointer to device name to open

int rfu
- reserved for future use

Returns: device handle on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Device Management

Mode: synchronous

Platform: DM3

Description

The **dcb_open( )** function opens a conference device and returns a unique handle to identify the device. The device may be a conference board or a DSP on the board. All subsequent references to the opened device must be made using the device handle. Refer to the *Audio Conferencing Programming Guide* for complete information about device names.

Parameter	Description
name	points to an ASCIIZ string that contains the name of a valid DSP device or board device
rfu	reserved for future use. Set this parameter to 0.

Notes: ***1.*** If a parent process opens a device and enables events, there is no guarantee that the child process will receive a particular event.

2. No action can be performed on a conference device until it is opened.

Cautions

This function fails when:

- The device name is invalid.
- The system has insufficient memory to complete the open.

Errors

The **dcb_open( )** function does not return errors in the standard return code format. If an error occurred during the **dcb_open( )** call, a -1 will be returned. If a call to **dcb_open( )** is successful, the return value will be a handle for the opened device.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

main()
{
   int bddevh;

   /* open board 1*/
   if ((bddevh = dcb_open("dcbB1", 0)) == -1) {
      printf("Cannot open device dcbB1: system error/n");
      exit(1);
   }
   else
      printf("Board %s is OPEN\n", ATDV_NAMEP(bddevh));

   /* Done processing - Close device */
   if (dcb_close(bddevh) == -1) {
      printf("Cannot close dcbB1: system error/n");
      exit(1);
   }
}
```

See Also

- dcb_close( )

dcb_remfromconf( )

Name: int dcb_remfromconf(devh, confid, cdt)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
MS_CDT *cdt	• pointer to conference descriptor element

Returns: 0 if success
-1 if failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

■ Description

The **dcb_remfromconf( )** function removes a conferee from a conference. The conference identifier is the value previously returned by the **dcb_estconf( )** function. In this case, the channel attributes of the MS_CDT structure are ignored.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference from which the conferee will be removed
cdt	points to an MS_CDT data structure that specifis the conferee (the attributes of the conferee are ignored)

Notes: ***1.*** Call the appropriate **xx_unlisten( )** function before removing the TDM bus time slot member.

2. Calling this function frees one conference resource.

■ Cautions

- An error will be returned if this function is used to attempt removal of the last remaining conferee from a conference. The **dcb_delconf( )** function must be used to end a conference.
- This function also fails when:
  - The device handle passed is invalid.
  - The conference identifier is invalid.
  - The conferee to be removed is not part of the specified conference.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   3

main()
{
   int  dspdevh;                    /* Conference descriptor table */
   int  confid;                     /* Conference identifier */
   int  tsdevh1, tsdevh2, tsdevh3;  /* DTI time slot device handles */
   long  scts;                      /* Transmit time slot */
   SC_TSINFO  tsinfo;               /* Time slot information structure */

   /* Open conference board 1, DSP 3 device */
   if ((dspdevh = dcb_open("dcbB1D3",0) == -1) {
      printf("Cannot open dcbB1D3: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
     printf( "Cannot open dtiB1T1: system error/n");
     exit(1);
   }

   /* Prepare TDM bus time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* Get transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message = %s",ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts;   /* SCbus time slot returned */
   cdt[0].chan_sel  = MSPN_TS;     /* ...by dt_getxmitslot() */
   cdt[0].chan_attr = MSPA_NULL;   /* Conferee has no special attributes */

   /* Open DTI board 1, time slot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf("Cannot open dtiB1T2: system error/n");
      exit(1);
   }
```

```
/* Get transmit time slot of DTI tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up CDT structure */
cdt[1].chan_num  = (int)scts;     /* TDM bus time slot returned */
cdt[1].chan_sel  = MSPN_TS;       /* ...from dt_getxmitslot() */
cdt[1].chan_attr = MSPA_TARIFF;   /* Conferee receives periodic tariff tone */

/* Open board 1, tslot 3 */
if ((tsdevh3 = dt_open("dtiB1T3",0)) == -1) {
   printf( "Cannot open dtiB1T3: system error/n");
   exit(1);
}

/* Get transmit time slot of DTI tsdevh3 */
if (dt_getxmitslot(tsdevh3, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh3));
   exit(1);
}

/* Set up CDT structure */
cdt[2].chan_num  = (int)scts;        /* TDM bus time slot returned */
cdt[2].chan_sel  = MSPN_TS;          /* ...from dt_getxmitslot() */
cdt[2].chan_attr = MSPA_TARIFF;      /* Conferee receives periodic tariff tone */

/* Establish 3 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for DTI tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for the DTI tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Do a listen for the DTI tsdevh3 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[2].chan_lts;

if (dt_listen(tsdevh3,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh3));
   exit(1);
}

/* Select tsdevh1 as conferee to remove from conference */
```

```
   /* Unlisten the listening device tsdevh1 */
   if (dt_unlisten(tsdevh1) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Prepare TDM bus time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* Get transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Prepare the MS_CDT structure */
   cdt[0].chan_num = (int)scts;
   cdt[0].chan_sel = MSPN_TS;

   /* And remove tsdevh1 from the conference */
   if (dcb_remfromconf(dspdevh, confid, &cdt[0]) == -1) {
      printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Unlisten the remaining listening time slots */
   if (dt_unlisten(tsdevh2) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   if (dt_unlisten(tsdevh3) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh3));
      exit(1);
   }

   /* Delete the conference */
   if(dcb_delconf(dspdevh, confid)) == -1) {
      printf("Cannot delete conference %d : Error Message = %s", confid,
               ATDV_ERRMSGP(dspdevh));
      exit(1);
   }

   /* Close all open devices */
   if (dt_close(tsdevh1) == -1){
      printf("Error closing tsdevh1\n");
      exit(1);
   }
   if (dt_close(tsdevh2) == -1){
      printf("Error closing tsdevh2\n");
      exit(1);
   }
   if (dt_close(tsdevh3) == -1){
      printf("Error closing tsdevh3\n");
      exit(1);
   }
   if (dcb_close(dspdevh) == -1){
      printf("Cannot close dcbB1D3: system error/n");
      exit(1);
   }
}
```

See Also

- dcb_addtoconf( )

- **dcb_delconf( )**
- **dcb_estconf( )**

dcb_setbrdparm( )

Name: int dcb_setbrdparm(devh, param, valuep)

Inputs:

int devh	• valid board device handle
unsigned char param	• device parameter defined name
void * valuep	• pointer to the parameter value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Configuration

Mode: synchronous

Platform: DM3

Description

The **dcb_setbrdparm( )** function sets conference board device parameters.

Parameter	Description
devh	specifies the valid device handle obtained when the board device was opened using **dcb_open( )**
param	indicates the parameter whose value is to be set
valuep	the address of the integer or MS_VOL structure containing the values to be assigned to the parameter

The valid values for **param** and **valuep** are shown below:

Note: For MSG_ACTID, MSG_ACTTALKERNOTIFYINTERVAL, and MSG_TONECLAMP, **valuep** points to an integer value. For MSG_VOLDIG, **valuep** points to an MS_VOL data structure.

MSG_ACTID (Active Talker Identification)

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

conference. Active talkers are determined by their loudness; i.e., the strength of their "non-silence" energy.

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

MSG_ACTTALKERNOTIFYINTERVAL (Active Talker Notification Interval)

Changes the interval specifying how frequently the Active Talker status is updated. The value is specified in 10 ms units. The default value is 100 (in 10 ms units), which results in a 1-second interval, and the maximum value is 1000, which results in a 10-second interval.

Note: If a low value is used, it can affect system performance due to the more frequent updating of the status (which results in a high quantity of internal notification messages). If a high value is used, it will result in less frequent updating of status, but the non-silence energy of a conferee may not be reported if it occurs between notification updates. For example, if the notification interval is set to 2 seconds and a conferee only says "yes" or "no" quickly in between notifications, that vocalization by the conferee will not be reported.

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 for all parties to reduce the amount of DTMF tones heard in a conference. Tone clamping applies to the transmitted audio going into the conference and does not affect DTMF function. It is meaningful only in the full duplex or the transmit-only mode. Possible values are TONECLAMP_ON or TONECLAMP_OFF. TONECLAMP_OFF is the default. (To enable on a per-party basis, set the MSPA_PARTY_TONECLAMP attribute in the MS_CDT structure for the party.) Even with tone clamping, 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.

Cautions

- All parameter values must be integers or MS_VOL data structures, but since this routine expects a void pointer to **valuep**, the address must be cast as a void*.
- This function fails when:
  - The device handle is invalid
  - The parameter specified is invalid

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

main()
{

   int  bddevh;                  /* Board dev descriptor variables */
   int  valuep = ACTID_ON;
   MS_VOL volume                 /*volume control */

   /* open DCB board 1 */
   if ( (bddevh = dcb_open("dcbB1", 0)) == -1) {
      printf("Cannot open device dcbB1: system error/n");
      exit(1);
   }

   /* Enable Active talker identification */
   if (dcb_setbrdparm(devh, MSG_ACTID, &valuep) == -1) {
      printf("Error setting board param:0x%x\n ",ATDV_LASTERR(devh));
      exit(1);
   }

   volume.vol_control = ON;
   volume.vol_up      = 2;
   volume.vol_reset   = 5;
   volume.vol_down    = 8;

   if (dcb_setbrdparm(devh, MSG_VOLDIG,(void *)&volume) == -1) {
      printf("Error getting board param:0x%x\n ", ATDV_LASTERR(devh));
      exit(1);
   }

   /*
    * Continue processing
    */

   /* Done processing - Close device */
   if ( dcb_close(bddevh) == -1) {
      printf("Cannot close device dcbB1: system error/n");
      exit(1);
   }
}
```

See Also

- dcb_getbrdparm( )
- dcb_GetPartyParm( )
- dcb_SetPartyParm( )

dcb_setcde( )

Name: int dcb_setcde(devh, confid, cdt)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
MS_CDT *cdt	• pointer to conference descriptor element

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_setcde( )** function changes the attributes of a conferee in an existing conference.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference identifier number
cdt	points to an MS_CDT data structure that defines the updated attributes of the conferee

Cautions

- The **dcb_setcde( )** function cannot be used to enable or disable echo cancellation. Once a party belongs to a conference, you cannot enable or disable echo cancellation for that party. If attempted, it has no effect. To enable echo cancellation, use **dcb_estconf( )** or **dcb_addtoconf( )**
- This function fails when:
  - The device handle specified is invalid.
  - The conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   2

main()
{
   int  dspdevh;               /* DSP device handle */
   MS_CDT  cdt[NUM_PARTIES];   /* Conference descriptor table */
   int  confid;                /* Conference identifier */
   int  tsdevh1, tsdevh2;      /* DTI time slot device handle */
   long  scts;                 /* TDM bus transmit time slot */
   SC_TSINFO  tsinfo;          /* Time slot information structure */

   /* Open conference board 1, DSP 3 device */
   if ((dspdevh = dcb_open("dcbB1D3",0)) == -1) {
      printf("Cannot open dcbB1D3: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Prepare time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* get transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts;     /* TDM bus time slot returned */
   cdt[0].chan_sel  = MSPN_TS;       /* by dt_getxmitslot() */
   cdt[0].chan_attr = MSPA_TARIFF;   /* Conferee will receive period tariff tones */

   /* Open DTI board 1, time slot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf( "Cannot open dtiB1T2: system error/n");
      exit(1);
   }

   /* Get transmit time slot of DTI tsdevh2 */
   if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
      printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[1].chan_num  = (int)scts;    /* TDM bus time slot returned */
   cdt[1].chan_sel  = MSPN_TS;      /* returned from getxmitslot */
   cdt[1].chan_attr = MSPA_PUPIL;   /* Conferee may be coached later */
```

```
/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Now change the attribute of the last added conferee */
/* NOTE : scts still contains the transmit time slot of tsdevh2 */
cdt[0].chan_num = (int)scts;
cdt[0].chan_sel = MSPN_TS;
cdt[0].chan_attr = MSPA_TARIFF;

if((dcb_setcde(dspdevh, confid, &cdt[0])) == -1) {
  printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
  exit(1);
}

/* Perform 'unlistens' on the listening DTI time slots */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Delete the conference */
if(dcb_delconf(dspdevh, confid) == -1) {
   printf("Cannot delete conference %d. Error Message = %s", confid,
           ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Close all open devices */
if (dt_close(tsdevh1) == -1){
   printf("Error closing tsdevh1\n");
   exit(1);
}

if (dt_close(tsdevh2) == -1){
   printf("Error closing tsdevh2\n");
   exit(1);
}
```

```
    if (dcb_close(dspdevh) == -1){
       printf("Cannot close dcbB1D3: system error/n");
       exit(1);
    }

}
```

See Also

- dcb_addtoconf( )
- dcb_estconf( )
- dcb_getcde( )

dcb_setdigitmsk( )

Name: int dcb_setdigitmsk(devh, confid, bitmask, action)

Inputs:

int devh	• valid DSP device handle
int confid	• conference identifier
unsigned short bitmask	• event bitmask
int action	• change type

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Configuration

Mode: synchronous

Platform: DM3

Description

The **dcb_setdigitmsk( )** function enables specific digit detection for a conference. The current bitmask is examined by a call to **dcb_getdigitmsk( )**.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference identifier number
bitmask	indicates the DTMF digit detection bitmask
action	specifies how the digit mask is changed. Possible values are: • CBA_ADDMSK – enables messages from the conference specified in **bitmask**, in addition to previously set events. • CBA_SETMSK – enables notification of events specified in **bitmask** and disables notification of previously set events. • CBA_SUBMSK – disables messages from the conference specified in **bitmask**.

Note: If MSG_VOLDIG is enabled to give transparent volume control to the conferees, the digits for volume increase, decrease, and reset will not cause digit events to be generated. As a result, the application will not know if the volume changes.

The **bitmask** determines the digits to be detected. Upon detection of a DTMF digit, a DCBEV_DIGIT event is generated on a DSP device handle. The **sr_getevtdatap( )** function can be used to retrieve the DCB_DIGITS data structure.

The possible values for **bitmask** are:

CBMM_ZERO
Detect digit 0

CBMM_ONE
Detect digit 1

CBMM_TWO
Detect digit 2

CBMM_THREE
Detect digit 3

CBMM_FOUR
Detect digit 4

CBMM_FIVE
Detect digit 5

CBMM_SIX
Detect digit 6

CBMM_SEVEN
Detect digit 7

CBMM_EIGHT
Detect digit 8

CBMM_NINE
Detect digit 9

CBMM_STAR
Detect digit *

CBMM_POUND
Detect digit #(octothorpe)

CBMM_A
Detect digit A

CBMM_B
Detect digit B

CBMM_C
Detect digit C

CBMM_D
Detect digit D

CBMM_ALL
Detect ALL digits

For example, to enable notification of the digits specified in the **bitmask** parameter and disable notification of previously set digits:

- specify the digits to enable in the **bitmask** field
- specify the CBA_SETMSK in the **action** field

To enable an additional digit specified in **bitmask** without disabling the currently enabled digits:

- specify the digits in **bitmask**
- specify CBA_ADDMSK in the **action** field

To disable digits in **bitmask** without disabling any other digits:

- specify the digits in **bitmask**
- specify CBA_SUBMSK in the **action** field

To disable all currently enabled digits:

- specify 0 in **bitmask**
- specify CBA_SETMSK in the **action** field

Cautions

This function fails when:

- The device handle specified is invalid.
- The action specified is invalid.
- Invalid conference identifier.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define   NUM_PARTIES   2

main()
{
   int  dspdevh;               /* DSP device handle */
   int  confid;                /* Conference Identifier */
   unsigned int bitmask;       /* Digit bitmask */
   int  tsdevh1, tsdevh2;      /* DTI time slot device handles */
   MS_CDT  cdt[NUM_PARTIES];   /* Conference descriptor table */
   long scts;                  /* TDM bus transmit time slot */

   /* Open conference board 1, DSP 2 device */
   if ((dspdevh = dcb_open("dcbB1D2",0)) == -1) {
      printf("Cannot open dcbB1D2: system error/n");
      exit(1);
   }
```

```
/* Open DTI board 1, time slot 1 */
if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
   printf( "Cannot open dtiB1T1: system error/n");
   exit(1);
}

/* Prepare the time slot information structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;

/* Retrieve the TDM bus transmit time slot for tsdevh1 */
if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Set up the MS_CDT structure */
cdt[0].chan_num  = (int)scts;     /* TDM bus time slot returned */
cdt[0].chan_sel  = MSPN_TS;       /* by dt_getxmitslot() */
cdt[0].chan_attr = MSPA_TARIFF;   /* Conferee receives periodic tariff tones */

/* Open board 1, tslot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf( "Cannot open dtiB1T2: system error/n");
   exit(1);
}

/* Prepare the time slot information structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;

/* Retrieve the TDM bus transmit time slot for tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up the MS_CDT structure */
cdt[1].chan_num  = (int)scts;     /* TDM bus time slot returned */
cdt[1].chan_sel  = MSPN_TS;       /* by dt_getxmitslot() */
cdt[1].chan_attr = MSPA_TARIFF;   /* Conferee receives periodic tariff tones */

/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == 1) {
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for the DTI tsdevh1 device */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for the DTI tsdevh2 device */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
```

```
/*
 * Enable DTMF detection for digits 1,3,5 only */
 */
if (dcb_setdigitmsk(dspdevh, confid, CBMM_ONE|CBMM_THREE|CBMM_FIVE,
      CBA_SETMSK)) == -1) {
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/*
 *  Continue processing
 */

/* Perform 'unlistens' on all DTI listening time slots */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Delete the conference */
if(dcb_delconf(dspdevh, confid) == -1) {
   printf("Cannot delete conference %d. Error Message = %s", confid,
           ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* And close all open devices */
if (dt_close(tsdevh1) == -1){
   printf("Error closing tsdevh1\n");
   exit(1);
}

if (dt_close(tsdevh2) == -1){
   printf("Error closing tsdevh2\n");
   exit(1);
}

if(dcb_close(dspdevh) == -1) {
  printf("Cannot close dcbB1D2: system error/n");
  exit(1);
}
```

See Also

- **dcb_getdigitmsk( )**

dcb_SetPartyParm( )

Name: int dcb_SetPartyParm(hSrlDevice, nConferenceID, Partycdt, unPartyParm, Value, rfu)

Inputs:

int hSrlDevice	• valid DSP device handle
int nConferenceID	• conference identifier
MS_CDT *Partycdt	• pointer to conference descriptor table element
unsigned int unPartyParm	• the parameter whose value is to be altered
void *Value	• the address of the integer or structure containing the value(s) to be assigned
void *rfu	• void pointer reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3 (but not HMP)

Description

The **dcb_SetPartyParm( )** function sets the parameters for a conferee (conference party).

Note: This function is **not** supported on Intel NetStructure® Host Media Processing (HMP) software.

Parameter	Description
hSrlDevice	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
nConferenceID	specifies the conference identifier number
Partycdt	points to an MS_CDT data structure that defines the attributes of the conferee *Note:* The chan_attr field of the MS_CDT structure is not used and is ignored by the **dcb_SetPartyParm( )** function.
unPartyParm	specifies the parameter whose value is to be altered. See list of parameters following this table.
Value	points to the integer or structure containing the value to be assigned to **unPartyParm**. See list of valid values for a parameter following this table.
rfu	void pointer reserved for future use

The valid values for **unPartyParm** and **Value** are shown below:

MSPA_OUTPUTVOLABS
: Specifies the absolute output volume level. Output volume refers to the volume that a conferee hears from a conference. Valid values are: -24 dB to +12 dB, in increments of +/- 1 dB.

MSPA_OUTPUTVOLRELCUR
: Specifies the output volume level relative to the current volume level produced by the conference. Valid values are: -36 dB to +36 dB. The resulting volume should not exceed -24 dB to +12 dB.

Cautions

- This function is **not** supported on Intel NetStructure® Host Media Processing (HMP) software.
- This function fails when:
  - the device handle specified is invalid.
  - the conference identifier is invalid.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"
#define   NUM_PARTIES   2

main()
{
   int  dspdevh = -1;            /* DSP device handle */
   MS_CDT cdt[NUM_PARTIES]={0};  /* Conference descriptor table */
   int  confid = -1;             /* Conference ID */
   int  tsdevh1=-1, tsdevh2=-1;  /* DTI time slot device handle */
   long  scts = -1;              /* TDM bus transmit time slot */
   SC_TSINFO  tsinfo = {0};      /* Time slot information structure */
   int vol;                      /* Conferee volume level */

   /* Open conference board 1, DSP 3 device */
   if ((dspdevh = dcb_open("dcbB1D3",0)) == -1) {
      printf("Cannot open dcbB1D3 : system error/n");
      exit(1);
   }
   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }
   /* Prepare time slot information structure */
   tsinfo.sc_numts = 1;
```

```
tsinfo.sc_tsarrayp = &scts;
/* get transmit time slot of DTI tsdevh1 */
if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}
/* Set up CDT structure */
cdt[0].chan_num  = (int)scts;     /* TDM bus time slot returned */
cdt[0].chan_sel  = MSPN_TS;       /* by dt_getxmitslot() */

/* Open DTI board 1, time slot 2 */
if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
   printf( "Cannot open dtiB1T2 : system error/n");
   exit(1);
}
/* Get transmit time slot of DTI tsdevh2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message = %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Set up CDT structure */
cdt[1].chan_num  = (int)scts;    /* TDM bus time slot returned */
cdt[1].chan_sel  = MSPN_TS;      /* returned from getxmitslot */
/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message = %s",ATDV_ERRMSGP(dspdevh));
   exit(1);
}
/* Do a listen for tsdevh1 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;
if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for tsdevh2 */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;
if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Now change the volume for the 1st conferee */
/* NOTE : scts still contains the transmit time slot of tsdevh2 */
cdt[0].chan_num = (int)scts;
cdt[0].chan_sel = MSPN_TS;
   vol = -4;
if((dcb_SetPartyParm(dspdevh, confid, &cdt[0], MSPA_OUTPUTVOLABS, &vol, 0)) == -1) {
  printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
  exit(1);
}
/* Perform 'unlistens' on the listening DTI time slots */
if (dt_unlisten(tsdevh1) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}
if (dt_unlisten(tsdevh2) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}
/* Delete the conference */
if(dcb_delconf(dspdevh, confid) == -1) {
   printf("Cannot delete conference %d. Error Message = %s", confid,
           ATDV_ERRMSGP(dspdevh));
   exit(1);
}
```

```
   /* Close all open devices */
   if (dt_close(tsdevh1) == -1){
      printf("Error closing tsdevh1\n");
      exit(1);
   }
   if (dt_close(tsdevh2) == -1){
      printf("Error closing tsdevh2\n");
      exit(1);
   }
   if (dcb_close(dspdevh) == -1){
      printf("Cannot close dcbB1D3 : system error/n");
      exit(1);
   }

}
```

See Also

- **dcb_GetPartyParm( )**

dcb_unmonconf( )

Name: int dcb_unmonconf(devh, confid)

Inputs: int devh — • valid DSP device handle

int confid — • conference identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h
dcblib.h

Category: Conference Management

Mode: synchronous

Platform: DM3

Description

The **dcb_unmonconf( )** function removes a monitor from a conference.

Parameter	Description
devh	specifies the valid device handle obtained when the DSP device was opened using **dcb_open( )**
confid	specifies the conference identifier

Notes: ***1.*** Calling this function frees one resource.

2. Call the appropriate **xx_unlisten( )** function for each conferee listening to the monitored signal before **dcb_unmonconf( )** is called.

Cautions

This function fails when:

- The device handle specified is invalid.
- It is called for a non-conference board.
- An invalid conference is specified.
- A monitor does not exist in the conference.

Errors

If this function returns -1 to indicate failure, obtain the reason for the error by calling the Standard Runtime Library standard attribute function **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to retrieve either the error code or a pointer to the error description, respectively.

Refer to Chapter 5, "Error Codes" of this guide for a list of error codes. Error defines can be found in *dtilib.h, msilib.h* or *dcblib.h*.

Example

```
#include <windows.h>  /*include in Windows applications only; exclude in Linux*/
#include <stdio.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "dcblib.h"

#define NUM_PARTIES     2

main()
{
   int  dspdevh;                      /* DSP device handle */
   MS_CDT  cdt[NUM_PARTIES];          /* Conference descriptor table */
   int  confid;                       /* Conference identifier */
   int  tsdevh1, tsdevh2, tsdevh3;    /* DTI time slot device handles */
   long  lts, scts;                   /* listen/transmit time slots */
   SC_TSINFO  tsinfo;                 /* Time slot information structure */

   /* Open conference board 1, DSP 1 device */
   if ((dspdevh = dcb_open("dcbB1D1",0)) == -1) {
      printf("Cannot open dcbB1D1: system error/n");
      exit(1);
   }

   /* Open DTI board 1, time slot 1 */
   if ((tsdevh1 = dt_open("dtiB1T1",0)) == -1) {
      printf( "Cannot open dtiB1T1: system error/n");
      exit(1);
   }

   /* Prepare the time slot information structure */
   tsinfo.sc_numts = 1;
   tsinfo.sc_tsarrayp = &scts;

   /* Get transmit time slot of DTI tsdevh1 */
   if (dt_getxmitslot(tsdevh1, &tsinfo) == -1){
      printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
      exit(1);
   }

   /* Set up CDT structure */
   cdt[0].chan_num  = (int)scts;    /* TDM bus time slot returned */
   cdt[0].chan_sel  = MSPN_TS;      /* by dt_getxmitslot() */
   cdt[0].chan_attr = MSPA_NULL;    /* Conferee has no special attributes */

   /* Open DTI board 1, time slot 2 */
   if ((tsdevh2 = dt_open("dtiB1T2",0)) == -1) {
      printf( "Cannot open dtiB1T2: system error/n");
      exit(1);
   }

   /* Open board 1, time slot 3 */
   if ((tsdevh3 = dt_open("dtiB1T3",0)) == -1) {
      printf( "Cannot open dtiB1T3: system error/n");
      exit(1);
   }
```

```
/* get transmit time slot of DTI TS device 2 */
if (dt_getxmitslot(tsdevh2, &tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Set up CDT structure */
cdt[1].chan_num  = (int)scts;    /* TDM bus time slot returned */
cdt[1].chan_sel  = MSPN_TS;      /* by dt_getxmitslot() */
cdt[1].chan_attr = MSPA_NULL;    /* Conferee has no special attributes */

/* Establish a 2 party conference */
if (dcb_estconf(dspdevh, cdt, NUM_PARTIES, MSCA_ND, &confid) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Do a listen for the DTI tsdevh1 device */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdevh1,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
   exit(1);
}

/* Do a listen for the DTI tsdevh2 device */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdevh2,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
   exit(1);
}

/* Now monitor the conference on TDM bus time slot lts */
if((dcb_monconf(dspdevh, confid, &lts)) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}

/* Prepare a time slot info structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &lts;

/* And let a DTI time slot, tsdevh3, monitor the conference */
if (dt_listen(tsdevh3,&tsinfo) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh4));
   exit(1);
}

/* Perform an 'unlisten' for the DTI time slot */
if (dt_unlisten(tsdevh3) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(tsdevh4));
   exit(1);
}

/* Now remove the monitoring */
if((dcb_unmonconf(dspdevh,confid)) == -1){
   printf("Error Message : %s", ATDV_ERRMSGP(dspdevh));
   exit(1);
}
```

```
    /* Perform 'unlistens' for the remaining DTI time slots */
    if (dt_unlisten(tsdevh1) == -1){
       printf("Error Message : %s", ATDV_ERRMSGP(tsdevh1));
       exit(1);
    }
    if (dt_unlisten(tsdevh2) == -1){
       printf("Error Message : %s", ATDV_ERRMSGP(tsdevh2));
       exit(1);
    }

    /* Delete the conference */
    if(dcb_delconf(dspdevh, confid) == -1) {
       printf("Cannot delete conference %d : Error Message = %s", confid,
             ATDV_ERRMSGP(dspdevh));
       exit(1);
    }

    /* And close all open devices */
    if (dt_close(tsdevh1) == -1){
       printf("Error closing tsdevh1\n");
       exit(1);
    }
    if (dt_close(tsdevh2) == -1){
       printf("Error closing tsdevh2\n");
       exit(1);
    }
    if (dt_close(tsdevh3) == -1){
       printf("Error closing tsdevh3\n");
       exit(1);
    }
    if (dcb_close(dspdevh) == -1){
       printf("Cannot close dcbB1D1: system error/n");
       exit(1);
    }
}
```

See Also

- **dcb_estconf( )**
- **dcb_monconf( )**

Events 3

This chapter provides information on events that may be generated by the audio conferencing software.

An event indicates that a specific activity has occurred within a conference or conferences. For information on handling conference events, refer to the *Audio Conferencing API Programming Guide*. For details on event management and event handling, see the *Standard Runtime Library API Programming Guide* and *Standard Runtime Library API Library Reference*.

The following events may be generated by functions in the audio conferencing library:

DCBEV_BRIDGEESTABLISHED
: Returned by the **dcb_CreateBridge( )** function to indicate that a conference bridge has been established.

DCBEV_BRIDGEREMOVED
: Generated by the **dcb_DeleteBridge( )** to indicate that a conference bridge has been deleted.

DCBEV_CTU
: Generated when the conference descriptor table for a conferee has been updated.

DCBEV_DELALLCONF
: Returned by the **dcb_DeleteAllConferences( )** function to indicates that all active conferences have been successfully deleted.

DCBEV_DIGIT
: Returned when a digit detection event occurs.

  Note: Any conferee participating in a conference in receive-only mode (MSPA_MODERECVONLY attribute) cannot generate DTMF digits within the conference, therefore any digits dialed by a conferee in receive-only mode will not generate DCBEV_DIGIT events.

DCBEV_ERREVT
: Indicates an error has occurred within the application.

Data Structures 4

This chapter contains information about the data structures used by the audio conferencing API. The following data structures are used:

DCB_CT

```
typedef struct dcb_ct
{
      int confid;
      int chan_num;
      int chan_sel;
} DCB_CT;
```

Description

The DCB_CT data structure contains information about active talkers within a conference. Refer to the **dcb_gettalkers( )**, **dcb_GetAtiBitsEx( )**, and **dcb_setbrdparm( )** functions and to the *Audio Conferencing API Programming Guide* for more information about the active talker feature.

Field Descriptions

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

confid
: conference identifier number of the conference being monitored for active talkers

chan_num
: denotes the TDM bus transmit time slot number occupied by the active talker

chan_sel
: defines the specific meaning of the chan_num field. For the current System Software release, chan_sel must be set to the following value:
  - MSPN_TS: – TDM bus time slot

DCB_DIGITS

```
typedef struct dcb_digits
{
      unsigned char  dsp;
      int            confid;
      int            chan_num;
      int            chan_sel;
      int            chan_attr;
      unsigned char  digits[MAX_DCBDIGS+1];
      unsigned char  dig_type;
} DCB_DIGITS;
```

Description

The DCB_DIGITS data structure defines the format of DCBEV_DIGIT events that are generated when a conferee presses a pre-determined DTMF digit. The pre-determined DTMF digits are defined for a conference via the **bitmask** parameter in the **dcb_setdigitmsk( )** function.

Field Descriptions

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

dsp
: indicates the DSP of the conference that generated the event

confid
: specifies the conference identifier of the conference that generated the event

chan_num
: denotes the TDM bus transmit time slot number of the conferee that generated the event

chan_sel
: defines the specific meaning of the chan_num field. For the current System Software release, chan_sel must be set to the following value:
  - MSPN_TS: – TDM bus time slot

chan_attr
: describes the properties of the conferee that generated the event. Refer to the chan_attr field of the MS_CDT data structure for a list of valid conferee properties.

digits[MAX_DCBDIG+1]
: denotes an ASCIIZ string of detected DTMF digits

dig_type
: indicates the type of digit detected (DTMF)

MS_CDT

```
typedef struct ms_cdt
{
      int chan_num;     /*time slot number*/
      int chan_sel;     /*meaning of time slot number */
      int chan_attr;    /*attribute description*/
} MS_CDT;
```

Description

The MS_CDT data structure specifies a conferee and its attributes. The chan_attr field is a bitmask that contains the conferee's properties within the conference.

This data structure is used to define or retrieve a conferee and its attributes. The following functions specify this structure to *define* attributes: addtoconf, estconf, and setcde. The following functions specify this structure to *retrieve* attributes: getcde, getcnflist, and gettalkers. Also, the remfromconf function uses the data structure to specify a conferee to remove from the conference (in this case, the attributes are *ignored*).

Field Descriptions

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

chan_num
: denotes the TDM bus transmit time slot number of the device to be included in the conference

chan_sel
: defines the specific meaning of the chan_num field. For the current System Software release, chan_sel must be set to the following value:
  - MSPN_TS: – TDM bus time slot

chan_attr
: a bitmask that specifies the conferee's properties within the conference. The following attributes are described as setting ordefining the attribute; however, for functions that retrieve attributes, they would indicate whether the attribute is set. Valid settings are as follows:
  - MSPA_BROADCASTEN – Enables the broadcast feature for conferee. This parameter sets one party to talk while all others are muted.
  - MSPA_COACH – Estabilshes conferee as a coach. Coach is heard by the pupil only. (When using the coach/pupil feature on analog lines, make sure to use echo cancellation so that other parties do not hear an echo of the coach.)
  - MSPA_ECHOXCLEN – Enables echo cancellation for conferee. The echo cancellation feature supplies 128 tap (16 msec) echo cancellation with the audio conferencing interface. The default setting for echo cancellation is disabled. Echo cancellation can only be enabled when establishing the conference with **dcb_estconf( )** or when ading a party to the conference with **dcb_addtoconf( )**. It cannot be enabled using **dcb_setcde( )**, and if attempted, has no effect. Once a party belongs to a conference, you cannot enable or disable echo cancellation for that party. If a conferee/party resides on an analog line, make sure to use echo cancellation so that the other parties do not hear an echo.
  - MSPA_MODEFULLDUPLX – Establishes conferee in full duplex mode, in which conferee may transmit and receive in the conference. Conferee hears everyone except the

coach. This property is equivalent to MSPA_NULL and MSPA_MODENULL. See also "Notes" following the attributes.

- MSPA_MODENULL – Establishes conferee as a NULL party. Conferee has no dedicated transmit or receive slots. A NULL party is often used as a placeholder for establishing a conference for a fixed number of parties. This is equivalent to MSPA_NULL and MSPA_FULLDUPLX. See also "Notes" following the attributes.
- MSPA_MODERECVONLY – Establishes conferee in receive-only mode, in which conferee may only receive in the conference. 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. See also "Notes" following the attributes.
- MSPA_MODEXMITONLY – Establishes conferee in transmit-only mode, in which conferee may only transmit in the conference. See also "Notes" following the attributes.
- MSPA_NOAGC – Disables Automatic Gain Control for a conferee.
- MSPA_NULL – No special attributes for conferee. This is equivalent to MSPA_MODENULL and MSPA_FULLDUPLX.
- MSPA_PARTY_TONECLAMP – Enables DTMF tone clamping for the conferee. (To enable for all conferees, enable the MSG_TONECLAMP parameter with the **dcb_setbrdparm( )** function.) Even with tone clamping, 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.
- MSPA_PUPIL – Establishes conferee as a pupil. Pupil hears everyone including the coach. (When using the coach/pupil feature on analog lines, make sure to use echo cancellation so that other parties do not hear an echo of the coach.)
- MSPA_TARIFF – Enables periodic tone transmission to conferee so that conferee hears periodic tone for duration of call.

Notes: ***1.*** The MSPA_MODENULL, MSPA_MODERECVONLY, MSPA_MODEXMITONLY and MSPA_MODEFULLDUPLX attributes are mutually exclusive. Furthermore, the attributes cannot be ORed together with any other conference attributes when calling the **dcb_setcde( )** function. For example, to set a conferee's attributes as full-duplex with a tariff tone, you must call the **dcb_setcde( )** function twice: once to set the MSPA_MODEFULLDUPLX attribute and once to set the MSPA_TARIFF attribute.

2. The **dcb_setcde( )** function cannot be used to enable or disable echo cancellation. Once a party belongs to a conference, you cannot enable or disable echo cancellation for that party. If attempted, it has no effect. To enable echo cancellation, use **dcb_estconf( )** or **dcb_addtoconf( )**.

3. Only one coach and one pupil are allowed in a conference at any time. Specifying more than one of either will cause unexpected results.

4. If a conferee/party resides on an analog line, make sure to use echo cancellation so that the other parties do not hear an echo. This is especially important when using the coach/pupil feature on analog lines.

5. The default MSPA_NULL must be used if channel attributes are not specified.

6. Invalid attribute combinations may lead to unexpected results.

7. For a background music application, such as a dating chat line where two callers talk while music plays in the background, when the music resource is added to the conference, it should be added in transmit-only mode using MSPA_MODEXMITONLY. (A media load specifically for background music is typically used with these types of applications to achieve the right mix of

resources on the board and to maximize density. For HMP, the desired ratio of resources is determined by the license.)

MS_VOL

```
typedef struct ms_vol
{
      unsigned char vol_control;
      unsigned char vol_up;
      unsigned char vol_reset;
      unsigned char vol_down;
} MS_VOL;
```

Description

The MS_VOL data structure defines whether or not volume control is active for a conference and which DTMF digits increase, decrease and reset the volume. This data structure is used by the **dcb_setbrdparm( )** and **dcb_getbrdparm( )** functions to control volume on a board basis.

Note: To adjust output (or speaker) volume on an individual conferee basis, see the **dcb_SetPartyParm( )** and **dcb_GetPartyParm( )** functions.

Field Descriptions

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

vol_control
determines whether or not volume control is activated. Possible values are as follows:

- ON
- OFF

vol_up
indicates the DTMF digit used for increasing the volume level.

vol_reset
indicates the DTMF digit used to reset the volume to its default level.

vol_down
indicates the DTMF digit used for decreasing the volume level.

TS_BRIDGECDT

```
typedef struct bridgecdt
{
      MS_CDT cdtA;
      MS_CDT cdtB;
      unsigned int nBridgeID;
} TS_BRIDGECDT;
```

Description

The TS_BRIDGECDT data structure defines the two conferences that are included in a conference bridge and provides a unique identifier for the conference bridge. This data structure allows conferencing applications to maintain the timeslots associated with a conference bridge in one location.

The data structure is composed of three elements, two MS_CDT data structures that are used to transfer the conference bridge party timeslots to the application and an unsigned integer that defines a unique conference bridge identifier for asynchronous mode events.

Field Descriptions

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

cdtA
: conference descriptor element for the master conference

cdtB
: conference descriptor element for the conference that is bridged to the master conference

nBridgeID
: denotes the bridge identification number that uniquely identifies a conference bridge. This number is returned to the application by the **dcb_CreateBridge( )** function.

Error Codes 5

This chapter lists the error codes that may be returned by the audio conferencing functions.

The values of error codes that may be returned to the application by the conferencing devices are a subset of errors used by the Intel® Dialogic® Digital Network Interface products and the Intel® Dialogic® Modular Station Interface products. Error codes proceeded by EDT_ are taken from the Digital Network Interface library (*dtilib.h*) and error codes proceeded by E_ are taken from the Modular Station Interface library (*msilib.h*).

The following error codes can be generated by the audio conferencing API library:

EDT_ADDRS
Incorrect address.

EDT_BADBRDERR
Board is missing or defective.

EDT_BADCMDERR
Invalid or undefined command to driver.

EDT_BADCNT
Incorrect count of bytes requested.

EDT_BADDEV
Bad device error.

EDT_BADGLOB
Incorrect global parameter number.

EDT_BADPORT
First byte appeared on reserved port.

EDT_BADVAL
Invalid parameter value passed in value pointer.

EDT_CHKSUM
Incorrect checksum.

EDT_DATTO
Data reception timed out.

EDT_DTTSTMOD
In test mode; cannot set board mode.

EDT_FWERR
Firmware returned an error.

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

EDT_INVBD
Invalid board.

EDT_INVMSG
Invalid message.

EDT_INVTS
Invalid time slot.

EDT_MBFMT
Wrong number of bytes for multiple byte request.

EDT_MBIMM
Received an immediate termination.

EDT_MBINV
First byte appeared on data port.

EDT_MBOVR
Message was too long.

EDT_MBPORT
Received multiple byte data on port other than 0 or 1.

EDT_MBTERM
Terminating byte other than FEH or FFH.

EDT_MBUND
Under the number of bytes for a multibyte request.

EDT_MSGCNT
Count received did not match actual count.

EDT_NOCLK
No clock source present.

EDT_NOIDLEERR
Time slot is not in idle/closed state.

EDT_NOMEMERR
Cannot map or allocate memory in driver.

EDT_NOTDNLD
Not downloaded.

EDT_PARAMERR
Invalid parameter. This error occurs if you execute an audio conferencing library function on a board that does not support that particular function.

EDT_RANGEERR
Bad/overlapping physical memory range.

EDT_SH_BADINDX
Invalid Switching Handler index number.

EDT_SH_BADEXITS
Returned time slot is unsupported in current clock rate.

EDT_SH_BADLCLTS
Invalid local time slot number.

EDT_SH_BADMODE
Invalid bus mode.

EDT_SH_BADTYPE
Invalid local time slot type.

EDT_SH_LCLDSCNCT
Local time slot is already disconnected from TDM bus.

EDT_SH_LCLTSCNCT
Local time slot is already connected to the TDM bus.

EDT_SH_LIBBSY
Switching Handler Library is busy.

EDT_SH_LIBNOTINIT
Switching Handler Library has not been initialized.

EDT_SH_MISSING
Switching Handler is not present.

EDT_SH_NOCLK
Switching Handler Clock fallback failed.

EDT_SIGINS
Insertion signaling not enabled.

EDT_SIGTO
Transmit/receive did not update in time.

EDT_SIZEERR
Message too big or too small.

EDT_SKIPRPLYERR
A required reply was skipped.

EDT_STARTED
Cannot start when already started.

EDT_SYSTEM
Operating system error.

EDT_TMOERR
Timed out waiting for reply from firmware.

EDT_TSASN
Time slot already assigned.

E_MS1PTY
Cannot remove party from one party conference.

E_MSBADCHPARM
Invalid channel parameter number.

E_MSBADVAL
Invalid parameter value.

E_MSCHASNCNF
Channel is assigned to conference.

E_MSCNFFUL
Conference system is full.

E_MSCNFLMT
Exceeds conference limit.

E_MSGLOBREAD
Cannot read parameter globally.

E_MSINVCB
Invalid control block ID.

E_MSINVCATTR
Invalid conference attribute.

E_MSINVCNF
Invalid conference number.

E_MSINVDSP
Invalid DSP specified.

E_MSINVMT
Invalid multitasking function.

E_MSINVPATTR
Invalid party attribute.

E_MSINVPTYNUM
Invalid party number.

E_MSINVPTYCNT
Invalid number of parties specified.

E_MSINVPTYTYPE
Invalid conference member type.

E_MSINVVAL
Bad global parameter value.

E_MSINVTS
Invalid time slot number specified.

E_MSMONEXT
Monitor already exists for this conference.

E_MSNOCNF
No conferencing available on device.

E_MSNODSPTS
All time slots going to the DSP are busy.

E_MSNOFEMCH
No DCB/SC daughterboard to support this channel.

E_MSNOMON
No monitor exists for this conference.

E_MSNONCNFCH
Channel is not assigned to specified conference.

E_MSNOTS
No time slot assigned to channel.

E_MSPTYASN
Party already assigned.

E_MSSYSTEM
Operating system error.

E_MSTSASNCNF
Time slot already assigned to a conference.

Index