Conferencing API for Linux and Windows Operating Systems

Library Reference

November 2003

05-2066-001

Publication Date: November 2003

Document Number: 05-2066-001

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

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

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

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

Contents

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2066-001	November 2003	Initial version of document.

About This Publication

The following topics provide information about this *Conferencing API for Linux and Windows Operating Systems Library Reference*:

Purpose

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

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

Intended Audience

This information is intended for:

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

How to Use This Publication

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

The information in this document is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the various categories of conferencing functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to the conferencing functions.
- Chapter 3, "Events" provides an alphabetical reference to events that may be returned by the conferencing software.
- Chapter 4, "Data Structures" provides an alphabetical reference to the conferencing data structures.
- Chapter 5, "Error Codes" presents a list of error codes that may be returned by the conferencing software.

Related Information

See the following for more information:

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

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

Function Summary by Category 1

This chapter describes the categories into which the conferencing API library functions can be grouped.

1.1 Device Management Functions

Device management functions open and close devices.

cnf_Close( )
closes a physical board device

cnf_Open( )
opens a physical board device

1.2 Configuration Functions

Configuration functions allow you to alter, examine, and control the configuration of an open device.

cnf_EnableEvents( )
enables events for a physical board device or conference device

cnf_DisableEvents( )
disables events for a physical board device or conference device

cnf_GetBoardAttributes( )
gets attributes for a physical board device

cnf_GetConferenceAttributes( )
gets attributes for a conference device

cnf_GetPartyAttributes( )
gets attributes for a party

cnf_GetResourceCount( )
gets the count of resources available on the board

cnf_GetVolumeControl( )
gets DTMF digits used to control volume on a physical board device

cnf_SetBoardAttributes( )
sets attributes for a physical board device

cnf_SetConferenceAttributes( )
sets attributes for a conference device

cnf_SetPartyAttributes( )
sets attributes for a party

cnf_SetVolumeControl( )
sets DTMF digits used to control volume on a physical board device

1.3 Conference Management Functions

Conference management functions allow you create and delete conferences, add and remove parties to a conference, as well as create and delete conference bridges.

cnf_AttachConference( )
opens an existing conference

cnf_AddParty( )
adds a party to an existing conference

cnf_AddPartyByTimeslot( )
adds a third party to an existing conference

cnf_CreateBridge( )
bridges two existing conferences

cnf_CreateConference( )
creates a conference

cnf_DeleteAllConferences( )
deletes all conferences on a physical board device

cnf_DeleteBridge( )
deletes an existing bridge between two conferences

cnf_DeleteConference( )
deletes a conference

cnf_GetActiveTalkers( )
gets the list of active talkers in a conference

cnf_GetPartyList( )
gets the list of parties in a conference

cnf_RemoveParty( )
removes a party from an existing conference

Function Information 2

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

2.1 Function Syntax Conventions

The conferencing functions use the following syntax:

```
int cnf_function(device_handle, parameter1, ... parameterN, mode)
```

where:

int
: represents an integer return value that indicates if the function succeeded or failed. Possible values are:
  - CNF_SUCCESS if the function succeeds
  - CNF_FAILURE if the function fails

cnf_function
: represents the function name

device_handle
: represents the device handle, which is a numerical reference to a device, obtained when a device is opened. The device handle is used for all operations on that device. There are two types of device handles: physical board device handle and conference device handle.

parameter1
: represents the first parameter

parameterN
: represents the last parameter

mode
: indicates the mode of execution. Possible values are:
  - EV_ASYNC for asynchronous mode
  - EV_SYNC for synchronous mode

Note: The "Platform" row in the function reference page indicates which platforms (such as DM3 and Springware) are supported for each function. "DM3" refers to products based on the Intel® Dialogic® DM3 mediastream architecture. "Springware" refers to products based on earlier-generation architecture. Although the conferencing API is supported on DM3 boards only, the platform information is provided for consistency across all Intel Dialogic libraries.

cnf_AddParty( )

Name: int cnf_AddParty(a_nConfHandle, a_partyList, a_pUserContext, a_nMode)

Inputs:	int a_nConfHandle	• conference device handle
	TSPartyList *a_partyList	• pointer to party specific information
	void *a_pUserContext	• user-supplied pointer used in asynchronous mode
	int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_AddParty( )** function adds a party (conferee) to an already created conference. The device handle specifies the conference to which the party will be added. The TSPartyList structure contains information on the added party. When **cnf_AddParty( )** returns successfully, m_pPartyInfo[x].m_nPartyID will contain the party identification number.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_partyList	pointer to party specific information. For more information, see the data structure description for TSPartyList, on page 135 and TSPartyInfo, on page 134.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

- Successfully adding a party or conferee to a conference consumes a time slot on the TDM bus. To get a count of the resources available on the board, use **cnf_GetResourceCount( )**. The **m_nPartyCount** field in the TSPartyList structure specifies the number of parties for a given conference.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_PARTYADDED
All the parties added successfully

CNFEV_ADDPARTYFAILED
Add party failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int threadID;
     int index;
     int byTimeslot;
     int timeslot;
} strUserContext;

strUserContext g_confUserContext;
strUserContext g_partyUserContext;

int g_boardHandle = -1;
int g_confHandle = -1;
int g_partyID = -1;
char g_confName[20] = {0};
long evt_handler(unsigned long a_evtHandle);

int main()
{
     TSConferenceInfo conferenceInfo;
     TSConferenceAttribute conferenceAttribute;

     int voiceHandle = 0;
     TSPartyAttribute partyAttribute;
     TSPartyInfo partyInfo;
     TSPartyList partyList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }
```

```
if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
{
     printf("Failed to open brdB1\n");
     return (-1);
}

conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
conferenceAttribute.uValue.m_nVal = 1;

conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_confUserContext.threadID = GetCurrentThreadId();
g_confUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_confUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

voiceHandle = dx_open("dxxxB1C1", 0);
if (voiceHandle < 0)
{
     printf("Failed to open dxxxB1C1\n");
     /* ProcessError( ); */
     return (-1);
}

partyAttribute.m_nVersion = TSAttribute_VERSION_1;
partyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
partyAttribute.uValue.m_nVal = PARM_VAL_Party_Mode_FULL;

partyInfo.m_nVersion = TSPartyInfo_VERSION_1;
partyInfo.m_nHandle = voiceHandle;
partyInfo.m_nAttrCount = 1;
partyInfo.m_pAttrs = &partyAttribute;
partyInfo.m_nPartyID = -1;

partyList.m_nVersion = TSPartyList_VERSION_1;
partyList.m_nPartyCount = 1;
partyList.m_pPartyInfo = &partyInfo;

/* set up the user context */
g_partyUserContext.threadID = GetCurrentThreadId();
g_partyUserContext.index = 2;/* user's own number */
g_partyUserContext.byTimeslot = 0;
```

```
     if (cnf_AddParty(g_confHandle,
               &partyList,
               &g_partyUserContext,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_AddParty on conference handle %d", g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     /* Remove the party
     cnf_RemoveParty(…);
     */

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandle,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     int  byTimeslot;

     strUserContext           *pstrUserContext;
     TSPartyList              *pPartyListReply;
     void                     *pUserContext;
```

```
        eventType = sr_getevttype();                    /* check the event type */
        pEventData = sr_getevtdatap();                  /* get the event data p */
        pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
        pstrUserContext = (strUserContext*)(pUserContext);
        if (pstrUserContext != NULL)
        {
             /* Retrieve user context */
             threadID = pstrUserContext->threadID;
             index = pstrUserContext->index;
             byTimeslot = pstrUserContext->byTimeslot;
        }

        switch(eventType)
        {
             case CNFEV_PARTYADDED:

             printf("Got CNFEV_PARTYADDED event.\n");

             if (!byTimeslot)
             {
                  /* retrieve the reply */
                  pPartyListReply = (TSpartyList*)(pEventData);
                  g_partyID = pPartyListReply->m_pPartyInfo->m_nPartyID;
             }
             break;

             case CNFEV_ADDPARTYFAILED:
             printf("Got CNFEV_ADDPARTYFAILED event\n");
             printf("Error code = 0x%x, Error message = %s\n",\
                        ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

             if(!byTimeslot)
             {
                  /* retrieve the reply */
                  pPartyListReply = (TSpartyList*)(pEventData);
                  g_partyID = pPartyListReply->m_pPartyInfo->m_nPartyID;
             }
             break;

             /*
             :
             :
             */
             default:
             break;
        }

        return (0);
}
```

See Also

- **cnf_RemoveParty( )**
- **cnf_GetResourceCount( )**

cnf_AddPartyByTimeslot( )

Name: int cnf_AddPartyByTimeslot(a_nConfHandle, a_thirdPartyList, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle	• conference device handle
TSThirdPartyList *a_thirdPartyList	• pointer to information on third party
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_AddPartyByTimeslot( )** function adds a third party to an already created conference based on the party's transmit and receive time slots. The device handle specifies the conference to which the party will be added. The TSThirdPartyList structure contains information on the added party. When **cnf_AddPartyByTimeslot( )** returns successfully, m_pPartyInfo[x].m_nPartyID will contain the party identification number.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_thirdParyList	pointer to third party specific information. For more information, see the data structure description for TSThirdPartyList, on page 132 and TSThirdPartyInfo, on page 131.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

When using Intel telecom boards, it is recommended that you add a party to a conference via the **cnf_AddParty( )** function. Use **cnf_AddPartyByTimeslot( )** function to add a party on a non-Intel telecom board.

Cautions

- Successfully adding a party or conferee to a conference consumes a time slot on the TDM bus. To get a count of the resources available on the board, use **cnf_GetResourceCount( )**. The **m_nPartyCount** field in the TSThirdPartyList structure specifies the number of parties for a given conference.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_PARTYADDED
All the parties added successfully

CNFEV_ADDPARTYFAILED
Add party failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     int     byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confUserContext;
strUserContext   g_partyUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
int     g_partyID = -1;
char    g_confName[20] = {0};
long    evt_handler(unsigned long a_evtHandle);

int main()
{
     TSConferenceInfo conferenceInfo;
     TSConferenceAttribute conferenceAttribute;

     int voiceHandle = 0;
     TSPartyAttribute partyAttribute;
     TSThirdPartyInfo partyInfo;
     TSThirdPartyList partyList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);
```

```
if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
{
     printf("sr_enbhdlr failed to assign the handler\n");
     return (-1);
}

if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
{
     printf("Failed to open brdB1\n");
     return (-1);
}

conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
conferenceAttribute.uValue.m_nVal = 1;

conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_confUserContext.threadID = GetCurrentThreadId();
g_confUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_confUserContext,
          EV_ASYNC) == -1)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

/* Get transmit timeslot , Txmitslot, through custom timeslot mechanism

   Get receive timeslot , Rxmitslot, through custom timeslot mechanism
*/

partyAttribute.m_nVersion = TSAttribute_VERSION_1;
partyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
partyAttribute.uValue.m_nVal = PARM_VAL_Party_Mode_FULL;

partyInfo.m_nVersion = TSThirdPartyInfo_VERSION_1;
partyInfo.m_nAttrCount = 1;
partyInfo.m_pAttrs = &partyAttribute;
partyInfo.m_nPartyID = -1;
partyInfo.m_nTxTimeslot = Txmitslot;
partyInfo.m_nRxTimeslot = Rxmitslot;

partyList.m_nVersion = TSThirdPartyList_VERSION_1;
partyList.m_nPartyCount = 1;
partyList.m_pPartyInfo = &partyInfo;

/* set up the user context */
g_partyUserContext.threadID = GetCurrentThreadId();
g_partyUserContext.index = 2;
g_partyUserContext.byTimeslot = 1; /* By timeslot */
g_partyUserContext.partyHandle = voiceHandle;
```

```
     if (cnf_AddPartyByTimeslot(g_confHandle,
               &partyList,
               &g_partyUserContext,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_AddPartyByTimeslot on conference\
                    handle %d",   g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

          /* ProcessError( ); */
            return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     /* Remove the party

        cnf_RemoveParty(…);
     */

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandle,
               NULL,
               EV_ASYNC) == -1)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError( ); */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID ;
     int   partyHandle;
     int   index;
     int   eventDevice;
     char  *pDeviceName;
     int   byTimeslot;

     strUserContext           *pstrUserContext;
     TSThirdPartyList         *pTSThirdPartyListReply;
     void                     *pUserContext;
```

```
     eventType = sr_getevttype();                    /* check the event type */
     pEventData = sr_getevtdatap();                  /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if(pstrUserContext != NULL)
     {
          /* Retrieve user context */
          threadID = pstrUserContext->threadID;
          index = pstrUserContext->index;
          byTimeslot = pstrUserContext->byTimeslot;
          partyHandle = pstrUserContext->partyHandle;
     }

     switch(eventType)
     {
          case CNFEV_PARTYADDED:

          printf("Got CNFEV_PARTYADDED event.\n");
          if (byTimeslot)
          {
               /* retrieve the reply */
               pTSThirdPartyListReply = (TSThirdpartyList*)(pEventData);
               printf("Party ID: %d\n", \
               pTSThirdPartyListReply->m_pPartyInfo->m_nPartyID);
               g_partyID = pTSThirdPartyListReply->m_pPartyInfo->m_nPartyID;

               /* Listen to RxTimeslot
               *  ts = RxTimeslot;
               *  tsInfo.sc_numts = 1;
               *  tsInfo.sc_tsarrayp = &ts;
               *
               *  dx_listen(partyHandle, &tsInfo);
               */
          }
          break;

          case CNFEV_ADDPARTYFAILED:
          printf("Got CNFEV_ADDPARTYFAILED event\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
          if(byTimeslot)
          {
               pTSThirdPartyListReply = (TSThirdPartyList*)(pEventData);
               g_partyID = pTSThirdPartyListReply->m_PartyInfo->m_nPartyID;
          }
          break;

          /*
          :
          :
           */
          default:
          break;
     }
     return (0);
}
```

See Also

- cnf_RemoveParty( )

cnf_AttachConference( )

Name: int cnf_AttachConference(a_nBrdHandle, a_szConfName, a_confReply, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
char *a_szConfName	• pointer to conference name string
TSCreateConferenceReply *a_confReply	• pointer to conference information
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_AttachConference( )** opens an existing conference on a given physical board device. To open an existing conference in another process, the application needs to share the name across processes and specify the conference name as an input parameter. To create a new conference, the application needs to call **cnf_CreateConference( )**.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_szConfName	pointer to conference name in the format **cnfBnCy** (the conference ID consists of a hexadecimal number)
a_confReply	pointer to conference information returned such as conference name and conference handle. For more information, see TSCreateConferenceReply, on page 129. In asynchronous mode, specify NULL. The conference information will be returned with the event.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_CONFATTACHED
Conference opened successfully

CNFEV_ATTACHCONFFAILED
Conference open failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] ={0};

int main()
{
     /* The existing conference name passed in by another process */
     char *pAttachConfName = GetConferenceName();

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);
     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }
```

```
    /* set up the user context */
    g_strUserContext.threadID = GetCurrentThreadId();
    g_strUserContext.index = 1;     /* user's own number */

    if(cnf_AttachConference(g_boardHandle,
              pAttachConfName,
              NULL,
              &g_strUserContext,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if(cnf_DeleteConference(g_boardHandle,
              g_confHandle,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if( (cnf_Close(g_boardHandle)) == -1)
    {
         printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         return(-1);
    }
    return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
    PVOID pEventData;
    int   eventType;
    int   threadID;
    int   index;

    strUserContext           *pstrUserContext;
    TSCreateConferenceReply  *pTSCreateConfReply;
    void                     *pUserContext;

    eventType = sr_getevttype();                  /* check the event type */
    pEventData = sr_getevtdatap();                /* get the event data p */
    pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
    pstrUserContext = (strUserContext*)(pUserContext);
    if (pstrUserContext != NULL)
```

```
{
     /* Retrieve user context */
     threadID =   pstrUserContext->threadID;
     index =      pstrUserContext->index;
}

switch(eventType)
{
     case CNFEV_CONFATTACHED:

     printf("Got CNFEV_CONFATTACHED event.\n");

     /* retrieve the conference reply */
     pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

     printf("The Attached Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
     printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

     printf("Index from user context = %d\n", index);

     /* set conference handle */
     g_confHandle = pTSCreateConfReply->m_nConfHandle);

     /* set conference name */
     strcpy(g_confName, pTSCreateConfReply->m_szConfName);
     break;

     case CNFEV_ATTACHCONFFAILED:

     printf("Got CNFEV_ATTACHCONFFAILED event.\n");
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
     break;

     :
     :

     default:
     break;
}
     return (0);
}
```

See Also

- **cnf_DeleteConference( )**

cnf_Close( )

Name: int cnf_Close(a_nBrdHandle)

Inputs: int a_nBrdHandle
- physical board device handle

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous

Platform: DM3

Description

The **cnf_Close( )** function closes the physical board device opened previously in the same process using **cnf_Open( )**.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int g_BoardHandle = -1;
void main()
{
```

```
        if ((cnf_Open(&g_BoardHandle, "brdB1", 0)) == CNF_FAILURE)
        {
             printf("Failed to open board\n");
        }
        else
        {
             printf("Got board handle: %d\n", g_BoardHandle);
        }
        /* Perform other processing as needed */

        if ((cnf_Close(g_BoardHandle)) == CNF_FAILURE)
        {
             printf("Failed to close &s\n", ATDV_NAMEP(g_BoardHandle));
             printf("error code = 0x%x, error message = %s\n",
                       ATDV_LASTERR(g_BoardHandle), ATDV_ERRMSGP(g_BoardHandle));
        }
}
```

See Also

- **cnf_Open( )**

cnf_CreateBridge( )

Name: int cnf_CreateBridge(a_nConfHandle1, a_ nConfHandle2, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle1	• conference device handle for conference 1
int a_nConfHandle2	• conference device handle for conference 2
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_CreateBridge( )** function bridges two already created conferences specified by their respective device handles. In asynchronous mode, the completion event will be returned on the **a_nConfHandle1** device handle.

Parameters	Description
a_nConfHandle1	specifies the conference device handle used to identify conference 1
a_nConfHandle2	specifies the conference device handle used to identify conference 2
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Calling this function causes one party resource to be used from each board on which the conferences are created. If both of the conferences are created on the same board, two party resources on the same board are used. To get the count of available party resources, use **cnf_GetResourceCount( )**.

Cautions

The function returns success only if the specified bridge can be created between the two specified conferences. If not, the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_BRIDGECREATED
Two conferences bridged successfully

CNFEV_CREATEBRIDGEFAILED
Bridging two conferences failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confAUserContext;
strUserContext   g_confBUserContext;

int     g_boardHandle  = -1;
int     g_confHandleA = 0;
int     g_confHandleB= 0;
char    g_confNameA[20];
char    g_confNameB[20];

int main()
{
     /*The existing conference name, passed in from another process */
     char *pAttachConfNameA = GetConferenceNameA();
     char *pAttachConfNameB = GetConferenceNameB();

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }
```

```
/* set up the user context */
g_confAUserContext.threadID = GetCurrentThreadId();
g_confAUserContext.index = 1;/* user's number for Conf A */

if(cnf_AttachConference(g_boardHandle,
          pAttachConfNameA,
          NULL,
          &g_confAUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

/* set up the user context */
g_confBUserContext.threadID = GetCurrentThreadId();
g_confBUserContext.index = 2;/* user's number for Conf B */

if(cnf_AttachConference(g_boardHandle,
          pAttachConfNameB,
          NULL,
          &g_confBUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

if ((cnf_CreateBridge(g_confHandleA,
          g_confHandleB,
          NULL,
          EV_ASYNC)) == CNF_FAILURE)
{
     printf("Failed to issue cnf_ CreateBridge on conf handle %d, and %d\n", \
               g_confHandleA, g_confHandleB);
     printf("Error code = 0x%x, Error message = %s\n",
               ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

/* Delete the bridge
     :
     :
*/

/* wait for the event */
sr_waitevt(-1);
```

```
     if(cnf_DeleteConference(g_boardHandle,
               g_confHandleA,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandleA);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandleB,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandleB);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
          return(-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     void                     *pUserContext;

     eventType = sr_getevttype();               /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
           threadID =   pstrUserContext->threadID;
           index =      pstrUserContext->index;
     }
```

```
        switch(eventType)
        {
             case CNFEV_BRIDGECREATED:
             printf("Got CNFEV_BRIDGECREATED event.\n");
             break;

             case CNFEV_CREATEBRIDGEFAILED:
             printf("Got CNFEV_CREATEBRIDGEFAILED event.\n");
             printf("Error code = 0x%x, Error message = %s\n",
                       ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));
             break;

             case CNFEV_CONFATTACHED:
             print("Got CNFEV_CONFATTACHED event.\n");

             /* retrieve the conference reply */
             pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

             printf("The Attached Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
             printf("The Attached Conference Handle: %d\n",
                         pTSCreateConfReply->m_nConfHandle);

             printf("Index from user context = %d\n", index);

             /* set conference handle */
             if (index == 1)
             {
                  g_confHandleA = pTSCreateConfReply->m_nConfHandle);
                  /* set conference name */
                  strcpy(g_confNameA, pTSCreateConfReply->m_szConfName);
             }

             if (index == 2)
             {
                  g_confHandleB = pTSCreateConfReply->m_nConfHandle);
                  /* set conference name */
                  strcpy(g_confNameB, pTSCreateConfReply->m_szConfName);
             }
             break;

             case CNFEV_ATTACHCONFFAILED:
             printf("Got CNFEV_ATTACHCONFFAILED event.\n");
             printf("Error code = 0x%x, Error message = %s\n",
                       ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
             break;
                  :
                  :
             default:
             break;
        }
        return (0);
}
```

See Also

- **cnf_DeleteBridge( )**
- **cnf_GetResourceCount( )**

cnf_CreateConference( )

Name: int cnf_CreateConference(a_nBrdHandle, a_confInfo, a_confReply, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
TSConferenceInfo *a_confInfo	• pointer to conference information
TSCreateConferenceReply *a_confReply	• pointer to conference information
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_CreateConference( )** function creates a new conference on a given physical board device. To open an existing conference in another process, the application needs to share the conference name (contained in m_szConfName field of the TSCreateConferenceReply structure) across processes and call **cnf_AttachConference( )** from that process.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_confInfo	points to conference information contained in the TSConferenceInfo data structure. For more information, see TSConferenceInfo, on page 128.
a_confReply	points to conference information returned such as conference name and conference handle. For more information, see TSCreateConferenceReply, on page 129. In asynchronous mode, specify NULL. The conference information will be returned with the event.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_CONFCREATED
Conference created successfully

CNFEV_CREATECONFFAILED
Conference creation failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confUserContext;
strUserContext   g_partyUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] = {0};

int main()
{
     TSConferenceInfo           conferenceInfo;
     TSConferenceAttribute      conferenceAttribute;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }
```

```
     conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
     conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
     conferenceAttribute.uValue.m_nVal = 1;

     conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
     conferenceInfo.m_nPartyGroup = 0;
     conferenceInfo.m_nAttrCount = 1;
     conferenceInfo.m_pAttrs = &conferenceAttribute;

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;     /* user's own number */

     if(cnf_CreateConference(g_boardHandle,
               &conferenceInfo,
               NULL,
               &g_strUserContext,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandle,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }
     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     void                     *pUserContext;
```

```
        eventType = sr_getevttype();                    /* check the event type */
        pEventData = sr_getevtdatap();                  /* get the event data p */
        pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
        pstrUserContext = (strUserContext*)(pUserContext);

        if (pstrUserContext != NULL)
        {
             /* Retrieve user context */
             threadID =   pstrUserContext->threadID;
             index =      pstrUserContext->index;
        }

        switch(eventType)
        {
             case CNFEV_CONFCREATED:

             /* retrieve the conference reply */
             pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

             printf("The Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
             printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

             printf("Index from user context = %d\n", index);

             /* set conference handle */
             g_confHandle = pTSCreateConfReply->m_nConfHandle);

             /* set conference name */
             strcpy(g_confName, pTSCreateConfReply->m_szConfName);
             break;

             case CNFEV_CREATECONFFAILED:

             printf("Got CNFEV_CREATECONFFAILED event.\n");
             printf("Error code = 0x%x, Error message = %s\n",\
                           ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

             break;
                  :
                  :
             default:
             break;
        }
        return (0);
}
```

See Also

- **cnf_DeleteConference( )**

cnf_DeleteAllConferences( )

Name: int cnf_DeleteAllConferences(a_nBrdHandle, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_DeleteAllConferences( )** function deletes all conferences previously created using **cnf_CreateConference( )** or previously allocated using **cnf_AttachConference( )** on the given board device. After calling **cnf_DeleteAllConferences( )**, conference resources are de-allocated.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

The device handle previously returned by **cnf_CreateConference( )** or **cnf_AttachConference( )** is invalid after this function successfully completes.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_ALLCONFDELETED
All conferences deleted successfully

CNFEV_DELETEALLCONFFAILED
Delete all conferences operation failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] = {0};

int main()
{
     TSConferenceInfo conferenceInfo;
     TSConferenceAttribute conferenceAttribute;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
     conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
     conferenceAttribute.uValue.m_nVal = 1;

     conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
     conferenceInfo.m_nPartyGroup = 0;
     conferenceInfo.m_nAttrCount = 1;
     conferenceInfo.m_pAttrs = &conferenceAttribute;

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;     /* user's own number */
```

```
     if(cnf_CreateConference(g_boardHandle,
               &conferenceInfo,
               NULL,
               &g_strUserContext,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteAllConferences(g_boardHandle,
                NULL,
                EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteAllConferences \
                    on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",     \
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext            *pstrUserContext;
     TSCreateConferenceReply   *pTSCreateConfReply;
     void                      *pUserContext;

     eventType = sr_getevttype();                  /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if(pstrUserContext != NULL)
     {
           /* Retrieve user context */
           threadID =      pstrUserContext->threadID;
           index =      pstrUserContext->index;
     }
```

```
        switch(eventType)
        {
             case CNFEV_ALLCONFDELETED:
             printf("Got CNFEV_ALLCONFDELETED event.\n");
             break;

             case CNFEV_DELETEALLCONFFAILED:

             printf("Got CNFEV_DELETEALLCONFFAILED event\n");
             printf("Error code = 0x%x, Error message = %s\n", \
                       ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

             break;

                  :
                  :

             default:
             break;
        }

        return (0);
}
```

See Also

- **cnf_CreateConference( )**
- **cnf_DeleteConference( )**

cnf_DeleteBridge( )

Name: int cnf_DeleteBridge(a_nConfHandle1, a_ nConfHandle2, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle1	• conference device handle for Conference 1
int a_nConfHandle2	• conference device handle for Conference 2
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_DeleteBridge( )** function deletes an already created bridge between two conferences. The conferences are specified by their respective device handles. In asynchronous mode, the completion event is returned on the **a_nConfHandle1** device handle.

Parameters	Description
a_nConfHandle1	specifies the conference device handle used to identify conference 1
a_nConfHandle2	specifies the conference device handle used to identify conference 2
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Calling this function frees one party resource from each board on which the conferences are created. If both of the conferences were created on the same board, two party resources on the same board are freed.

Cautions

The function returns success only if the specified bridge between the two conferences can deleted. If not the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_BRIDGEDELETED
Bridge successfully deleted

CNFEV_DELETEBRIDGEFAILED
Deleting bridge failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confAUserContext;
strUserContext   g_confBUserContext;

int     g_boardHandle = -1;
int     g_confHandleA = -1;
int     g_confHandleB= -1;
char    g_confNameA[20] = {0};
char    g_confNameB[20] = {0};

int main()
{
     /*The existing conference name, passed in from another process */
     char *pAttachConfNameA = GetConferenceNameA();
     char *pAttachConfNameB = GetConferenceNameB();

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }
```

```
/* set up the user context */
g_confAUserContext.threadID = GetCurrentThreadId();
g_confAUserContext.index = 1;/* user's number for Conf A */

if(cnf_AttachConference(g_boardHandle,
          pAttachConfNameA,
          NULL,
          &g_confAUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

/* set up the user context */
g_confBUserContext.threadID = GetCurrentThreadId();
g_confBUserContext.index = 2; /* user's number for Conf B */

if(cnf_AttachConference(g_boardHandle,
          pAttachConfNameB,
          NULL,
          &g_confBUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

if ((cnf_CreateBridge(g_confHandleA,
          g_confHandleB,
          NULL,
          EV_ASYNC)) == CNF_FAILURE)
{
     printf("Failed to issue cnf_ CreateBridge on conf handle %d, and %d\n",\
               g_confHandleA, g_confHandleB);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);
```

```
     if ((cnf_DeleteBridge(g_ConfHandleA,
               g_ConfHandleB,
               NULL,
               EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_ DeleteBridge on conf handle %d, and %d\n",\
                    g_confHandleA, g_confHandleB);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandleA,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandleA);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandleB,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandleB);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return(-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
```

```
strUserContext           *pstrUserContext;
TSCreateConferenceReply  *pTSCreateConfReply;
void                     *pUserContext;

eventType = sr_getevttype();                    /* check the event type */
pEventData = sr_getevtdatap();                  /* get the event data p */
pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
pstrUserContext = (strUserContext*)(pUserContext);
if (pstrUserContext != NULL)
{
      /* Retrieve user context */
      threadID =      pstrUserContext->threadID;
      index =      pstrUserContext->index;
}

switch(eventType)
{
     case CNFEV_BRIDGECREATED:

     printf("Got CNFEV_BRIDGECREATED event.\n");
     break;

     case CNFEV_CREATEBRIDGEFAILED:

     printf("Got CNFEV_ CREATEBRIDGEFAILED event.\n");
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));
     break;

     case CNFEV_BRIDGEDELETED:

     printf("Got CNFEV_ BRIDGEDELETED event.\n");
     break;

     case CNFEV_DELETEBRIDGEFAILED:
     printf("Got CNFEV_DELETEBRIDGEFAILED event.\n");
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandleA), ATDV_ERRMSGP(g_confHandleA));
     break;

     case CNFEV_CONFATTACHED:

     printf("Got CNFEV_CONFATTACHED event.\n");

     /* retrieve the conference reply */
     pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

     printf("The Attached Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
     printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

     printf("Index from user context = %d\n", index);

     /* set conference handle */
     if (index == 1)
     {
          g_confHandleA = pTSCreateConfReply->m_nConfHandle);
          /* set conference name */
          strcpy(g_confNameA, pTSCreateConfReply->m_szConfName);
     }

     if (index == 2)
     {
          g_confHandleB = pTSCreateConfReply->m_nConfHandle);
          /* set conference name */
          strcpy(g_confNameB, pTSCreateConfReply->m_szConfName);
     }
     break;
```

```
            case CNFEV_ATTACHCONFFAILED:

            printf("Got CNFEV_ATTACHCONFFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

            break;
                 :
                 :

            default:
            break;
      }
      return (0);
}
```

See Also

- **cnf_CreateBridge( )**

cnf_DeleteConference( )

Name: int cnf_DeleteConference(a_nBrdHandle, a_nConfHandle, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
int a_nConfHandle	• conference device handle
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_DeleteConference( )** function deletes the conference previously created using **cnf_CreateConference( )** or attached using **cnf_AttachConference( )**. To delete a conference created in one process from another process, you must first open the same physical board device; then attach to the conference using **cnf_AttachConference( )** and specify the conference name to get a valid device handle; and finally delete the conference using **cnf_DeleteConference( )**.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

In asynchronous mode, the reply will contain the conference handle.

Cautions

If you delete a conference that is part of a bridge, you will break the bridge itself.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_CONFDELETED
Conference deleted successfully

CNFEV_DELETECONFFAILED
Delete conference operation failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] = {0};

int main()
{
     TSConferenceInfo conferenceInfo;
     TSConferenceAttribute conferenceAttribute;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);
     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
     conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
     conferenceAttribute.uValue.m_nVal = 1;
```

```
     conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
     conferenceInfo.m_nPartyGroup = 0;
     conferenceInfo.m_nAttrCount = 1;
     conferenceInfo.m_pAttrs = &conferenceAttribute;

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;     /* user's own number */

     if(cnf_CreateConference(g_boardHandle,
               &conferenceInfo,
               NULL,
               &g_strUserContext,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if(cnf_DeleteConference(g_boardHandle,
               g_confHandle,
               NULL,
               EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                     ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     void                     *pUserContext;
```

```
        eventType = sr_getevttype();                    /* check the event type */
        pEventData = sr_getevtdatap();                  /* get the event data p */
        pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
        pstrUserContext = (strUserContext*)(pUserContext);
        if(pstrUserContext != NULL)
        {
              /* Retrieve user context */
             threadID =   pstrUserContext->threadID;
             index =      pstrUserContext->index;
        }

        switch(eventType)
        {
             case CNFEV_CONFDELETED:
             printf("Got CNFEV_CONFDELETED event.\n");
             break;

             case CNFEV_DELETECONFFAILED:

             printf("Got CNFEV_DELETECONFFAILED event.\n");
             printf("Error code = 0x%x, Error message = %s\n",\
                       ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
             break;

                  :
                  :
             default:
             break;
        }

        return (0);
}
```

See Also

- **cnf_CreateConference( )**

cnf_DisableEvents( )

Name: int cnf_DisableEvents(a_nHandle, a_pEvents, a_pUserContext, a_nMode)

Inputs:

int a_nHandle	• handle for the board device or conference device
TSEventsList * a_pEvents	• pointer to list of events
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_DisableEvents( )** function disables the given set of events for a physical board device or conference device. The list of events being disabled should be applicable to the type of device being used. For example, DTMF events can be disabled only on a conferencing device.

Parameters	Description
a_nHandle	specifies the valid handle for a physical board device or conference device
a_pEvents	points to a list of events contained in the TSEventsList data structure. For more information on this structure, see TSEventsList, on page 130.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

The function returns success only if all the specified events can be disabled. If not, the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_EVTDISABLED
Events disabled successfully

CNFEV_DISABLEEVTFAILED
Disable events failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int g_boardHandle = -1;
long evt_handler(unsigned long a_evtHandle);

int main()
{
     EListOfEvents eventToDisable;
     TSEventsList  eventList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     eventToEnable = CNFENB_Event_StatusChanged;

     eventList.m_nVersion = TSEventsList_VERSION_1;
     eventList.m_nEventCount = 1;
     eventList.m_pEvents = &eventToDisable;

     if (cnf_DisableEvents(g_boardHandle,
                &eventList,
                NULL,
                EV_ASYNC) == CNF_FAILURE)
     {
           printf("Failed to issue cnf_DisableEvents on board handle %d", g_boardHandle);
           printf("Error code: %d, Error Message: %s",\ ATDV_NAMEP(g_boardHandle),
                     ATDV_ERRMSGP(g_boardHandle));
```

```
            /* ProcessError() */
            return (-1);
        }

        /* wait for the event */
        sr_waitevt(-1);

        if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
        {
            printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
            printf("Error code = 0x%x, Error message = %s\n",\
                        ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            return (-1);
        }

        return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
        int   eventType;

        eventType = sr_getevttype();                /* check the event type */

        switch(eventType)
        {
            case CNFEV_EVTDISABLED:

            printf("Got CNFEV_EVTDISABLED event.\n");
            break;

            case CNFEV_DISABLEEVTFAILED:

            printf("Got CNFEV_DISABLEEVTFAILED event.\n");
            printf("Error code: %d, Error Message: %s",\ ATDV_NAMEP(g_boardHandle),
                        ATDV_ERRMSGP(g_boardHandle));

            break;

            :
            :
            default:
            break;
        }

        return (0);
}
```

See Also

- **cnf_EnableEvents( )**

cnf_EnableEvents( )

Name: int cnf_EnableEvents(a_nHandle, a_pEvents, a_pUserContext, a_nMode)

Inputs:

int a_nHandle	• handle for the board device or conference device
TSEventsList * a_pEvents	• pointer to list of events
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_EnableEvents( )** function enables the given set of events for a physical board device or conference device. The list of events being enabled should be applicable to the type of device being used. For example, DTMF events can be enabled only on a conferencing device.

Parameters	Description
a_nHandle	specifies the valid handle for a physical board device or conference device
a_pEvents	points to a list of events contained in the TSEventsList data structure. For more information on this structure, see TSEventsList, on page 130.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

The function returns success only if all the specified events can be enabled. If not, the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_EVTENABLED
Events enabled successfully

CNFEV_ENABLEEVTFAILED
Enable events failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int g_boardHandle = -1;
long evt_handler(unsigned long a_evtHandle);

int main()
{
     EListOfEvents eventToEnable;
     TSEventsList  eventList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     eventToEnable = CNFENB_Event_StatusChanged;

     eventList.m_nVersion = TSEventsList_VERSION_1;
     eventList.m_nEventCount = 1;
     eventList.m_pEvents = &eventToEnable;

     if (cnf_EnableEvents(g_boardHandle,
                &eventList,
                NULL,
                EV_ASYNC) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_EnableEvents on board handle %d", g_boardHandle);
          printf("Error code: %d, Error Message: %s",\ ATDV_NAMEP(g_boardHandle),
                     ATDV_ERRMSGP(g_boardHandle));
```

```
     /* ProcessError() */
     return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     int   eventType;

     eventType = sr_getevttype();               /* check the event type */

     switch(eventType)
     {
          case CNFEV_EVTENABLED:

          printf("Got CNFEV_EVTENABLED event.\n");
          break;

          case CNFEV_ENABLEEVTFAILED:

          printf("Failed to issue cnf_EnableEvents on board handle %d", g_boardHandle);
          printf("Error code: %d, Error Message: %s",\ ATDV_NAMEP(g_boardHandle),
                     ATDV_ERRMSGP(g_boardHandle));

          break;

          :
          :
          default:
          break;
     }

     return (0);
}
```

See Also

- **cnf_DisableEvents( )**

cnf_GetActiveTalkers( )

Name: int cnf_GetActiveTalkers(a_nConfHandle, a_partyList, a_pUserContext, a_nMode)

Inputs:	int a_nConfHandle	• conference device handle
	TSPartyList *a_partyList	• pointer to list of active talkers in a conference
	void *a_pUserContext	• user-supplied pointer used in asynchronous mode
	int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetActiveTalkers( )** function returns the list of active talkers in a conference.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_partyList	points to the list of active talkers in a conference. For more information, see the data structure description for TSPartyList, on page 135 and TSPartyInfo, on page 134. In asynchronous mode, specify NULL. The list of talkers will be part of the event data.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Continuous Speech Processing API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

In synchronous mode, the application is responsible for allocating sufficient memory for the TSPartyList structure.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_ACTTALKERS
Get Active Talkers completed successfully

CNFEV_GETACTTALKERSFAILED
Get Active Talkers failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] = {0};

int main()
{
     /* The existing conference name passed in by another process */
     char *pAttachConfName = GetConferenceName();

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1; /* user's own number */
```

```
    if(cnf_AttachConference(g_boardHandle,
              pAttachConfName,
              NULL,
              &g_strUserContext,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if (cnf_GetActiveTalkers(g_confHandle,
              NULL,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_GetActiveTalkers \
                   on conference handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if(cnf_DeleteConference(g_boardHandle,
              g_confHandle,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
    {
         printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         return(-1);
    }

    return (0);
}
```

```
long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID = 0;
     int   index;
     int   eventDevice;

     strUserContext             *pstrUserContext;
     TSCreateConferenceReply    *pTSCreateConfReply;
     TSPartyList                *pPartyListReply;
     void                       *pUserContext;

     eventType = sr_getevttype();                   /* check the event type */
     pEventData = sr_getevtdatap();                 /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
          threadID =      pstrUserContext->threadID;
          index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_ACTTALKERS:

          printf("Got CNFEV_ACTTALKERS event.\n");

          /* retrieve the reply */
          pPartyListReply = (TSpartyList*)(pEventData);

          printf("Number of active talkers: %d\n", \
          pPartyListReply->m_nPartyCount);

          int numofActiveTalkers = pPartyListReply->m_nPartyCount;
          for (int count = 0; count< numofActiveTalkers; count++)
          {
          printf("Party ID: %d\n", pPartyListReply->m_pPartyInfo->m_nPartyID);

          pPartyListReply->m_pPartyInfo++;
          }
          break;

          case CNFEV_GETACTTALKERSFAILED:
          printf("Got CNFEV_GETACTTALKERSFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
          break;

          case CNFEV_CONFATTACHED:

          printf("Got CNFEV_CONFATTACHED event.\n");

          /* retrieve the conference reply */
          pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

          printf("The Attached Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
          printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

          printf("Index from user context = %d\n", index);

          /* set conference handle */
          g_confHandle = pTSCreateConfReply->m_nConfHandle);
```

```
            /* set conference name */
            strcpy(g_confName, pTSCreateConfReply->m_szConfName);
            break;

            case CNFEV_ATTACHCONFFAILED:

            printf("Got CNFEV_ATTACHCONFFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            break;

                 :
                 :

            default:
            break;
      }
      return (0);
}
```

See Also

- **cnf_GetPartyList( )**

cnf_GetBoardAttributes( )

Name: int cnf_GetBoardAttributes(a_nBrdHandle, a_pAttributes, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
TSAttributesList * a_pAttributes	• pointer to list of attributes
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetBoardAttributes( )** function gets the values of one or more physical board attributes.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pAttributes	points to a list of attributes contained in TSAttributesList structure. For more information, see TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_BRDATTR
Get board attributes completed successfully

CNFEV_GETBRDATTRFAILED
Get board attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int     g_boardHandle = -1;
main()
{
     TSBoardAttribute      getBoardAttribute;
     TSAttributesList      getAttributesList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&BoardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
     }
     else
     {
          printf("Got board Handle: %d\n", g_boardHandle);
     }

     getBoardAttribute.m_nVersion = TSAttribute_VERSION_1;
     getBoardAttribute.m_nAttrType = BOARD_Parm_Act_Talk_En;

     getAttributesList.m_nVersion = TSAttributesList_VERSION_1;
     getAttributesList.m_nAttrCount = 1;
     getAttributesList.m_pAttrs = &getBoardAttribute;

     if ((cnf_GetBoardAttributes(BoardHandle,
               &getAttributesList,
               NULL,
               EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_GetBoardAttributes on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);
```

```
     if( (cnf_Close(BoardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(BoardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(BoardHandle), ATDV_ERRMSGP(BoardHandle));
     }

}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext           *pstrUserContext;
     TSAttributesList         *pAttributesListReply;
     void                     *pUserContext;

     eventType = sr_getevttype();               /* check the event type */
     pEventData = sr_getevtdatap();             /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);

if (pstrUserContext != NULL)
     {
          /* Retrieve user context */
          threadID =      pstrUserContext->threadID;
          index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_BRDATTR:

          printf("Got CNFEV_BRDATTR event.\n");

          /* retrieve the reply */
          pAttributesListReply = (TSAttributesList*)(pEventData);

          for(int count=0;
               count < pAttributesListReply->m_nAttrCount;
               count++)
          {
               printf("Board Attribute Type: 0x%x\n", \
                         pAttributesListReply->m_pAttrs->m_nAttrType);
               printf("Board Attribute Value: %d\n",
               pAttributesListReply->m_pAttrs->uValue.m_nVal);

               pAttributesListReply->m_pAttrs++;
          }
          break;

          case CNFEV_GETBRDATTRFAILED:
          printf("Got CNFEV_GETBRDATTRFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          break;
               :
               :
          default:
          break;
     }
     return (0);
}
```

See Also

- cnf_SetBoardAttributes( )

cnf_GetConferenceAttributes( )

Name: int cnf_GetConferenceAttributes(a_nConfHandle, a_pAttributes, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle	• conference device handle
TSAttributesList *a_pAttributes	• pointer to list of attributes
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetConferenceAttributes( )** function gets the values of one or more conference attributes.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_pAttributes	specifies a list of attributes. For more information, see the data structure description for TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_CNFATTR
Get conference attributes completed successfully

CNFEV_GETCNFATTRFAILED
Get conference attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle =  -1;
char    g_confName[20];
long evt_handler(unsigned long a_evtHandle);

int main()
{
     TSConferenceInfo           conferenceInfo;
     TSConferenceAttribute      conferenceAttribute;

     TSConferenceAttribute      setConfAttribute;
     TSAttributesList           setAttributesList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
     conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
     conferenceAttribute.uValue.m_nVal = 1;

     conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
     conferenceInfo.m_nPartyGroup = 0;
     conferenceInfo.m_nAttrCount = 1;
     conferenceInfo.m_pAttrs = &conferenceAttribute;
```

```
/* set up the user context */
g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_strUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

getConfAttribute.m_nVersion = TSAttribute_VERSION_1;
getConfAttribute.m_nAttrType = CONF_Parm_ToneClamping;

getAttributesList.m_nVersion = TSAttributesList_VERSION_1;
getAttributesList.m_nAttrCount = 1;
getAttributesList.m_pAttrs = &getConfAttribute;

if ((cnf_GetConferenceAttributes(g_confHandle,
          &getAttributesList,
          NULL,
          EV_ASYNC)) == CNF_FAILURE)
{
     printf("Failed to issue cnf_ GetConferenceAttributes \
               on conference handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

if(cnf_DeleteConference(g_boardHandle,
          g_confHandle,
          &g_strUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
{
     printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
```

```
            return (-1);
      }

      return (0);

}

long evt_handler(unsigned long a_evtHandle)
{
      PVOID pEventData;
      int   eventType;
      int   threadID;
      int   index;

      strUserContext           *pstrUserContext;
      TSCreateConferenceReply  *pTSCreateConfReply;
      TSAttributesList         *pTSAttributesListReply;
      void                     *pUserContext;

      eventType = sr_getevttype();                  /* check the event type */
      pEventData = sr_getevtdatap();                /* get the event data p */
      pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
      pstrUserContext = (strUserContext*)(pUserContext);

      if (pstrUserContext != NULL)
      {
            /* Retrieve user context */
            threadID =      pstrUserContext->threadID;
            index =      pstrUserContext->index;
      }

      switch(eventType)
      {
            case CNFEV_CONFATTR:

            printf("Got CNFEV_CONFATTR event. \n");

            /* retrieve the reply */
            pTSAttributesListReply = (TSAttributesList*)(pEventData);

            for(int count=0;
                  count< pTSAttributesListReply->m_nAttrCount;
                  count++)
            {
                  printf("Conference Attribute Type: 0x%x\n",\
                              pTSAttributesListReply->m_pAttrs->m_nAttrType);

                  printf("Conference Attribute Value: %d\n",\
                              pTSAttributesListReply->m_pAttrs->uValue.m_nVal);

                  pTSAttributesListReply->m_pAttrs++;
            }
            break;

            case CNFEV_GETCONFATTRFAILED:

            printf("Got CNFEV_GETCONFATTRFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

            break;

            case CNFEV_CONFCREATED:
```

```
            printf("Got CNFEV_CONFCREATED event.\n");
            /* retrieve the conference reply */
            pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);
            printf("The Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
            printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);
            printf("Index from user context = %d\n", index);

            /* set conference handle */
            g_confHandle = pTSCreateConfReply->m_nConfHandle);

            /* set conference name */
            strcpy(g_confName, pTSCreateConfReply->m_szConfName);

            break;

            case CNFEV_CREATECONFFAILED:

            printf("Got CNFEV_CREATECONFFAILED event\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                     ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            break;
                 :
                 :
            default:
            break;
        }
        return (0);
}
```

See Also

- **cnf_SetConferenceAttributes( )**

cnf_GetPartyAttributes( )

Name: int cnf_GetPartyAttributes(a_nConfHandle, a_ partyID, a_pAttributes, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle	• conference device handle
int a_partyID	• party ID
TSAttributesList * a_pAttributes	• pointer to list of attributes
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.hh

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetPartyAttributes( )** function gets the values of one or more party attributes. In asynchronous mode, the event data will contain the party ID.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_partyID	specifies the party identifier returned by **cnf_AddParty( )** or **cnf_AddPartyByTimeslot( )**
a_pAttributes	specifies a list of attributes. For more information, see the data structure description for for TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Events

CNFEV_PARTYATTR
Get party attributes completed successfully

CNFEV_GETPARTYATTRFAILED
Get party attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confUserContext;
strUserContext   g_partyUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
int     g_partyID = -1;
char     g_confName[20] = {0};
long evt_handler(unsigned long a_evtHandle);

int main()
{
     int voiceHandle = 0;

     TSConferenceInfo           conferenceInfo;
     TSConferenceAttribute      conferenceAttribute;

     TSPartyAttribute           partyAttribute;
     TSPartyInfo                partyInfo;
     TSPartyList                partyList;

     TSPartyAttribute      getPartyAttribute;
     TSAttributesList      getPartyAttributeList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }
```

```
if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
{
     printf("Failed to open brdB1\n");
     return (-1);
}

conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
conferenceAttribute.uValue.m_nVal = 1;

conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_confUserContext.threadID = GetCurrentThreadId();
g_confUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_confUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

voiceHandle = dx_open("dxxxB1C1", 0);
if (voiceHandle < 0)
{
     printf("Failed to open dxxxB1C1\n");

     /* ProcessError( ); */
     return (-1);
}

partyAttribute.m_nVersion = TSAttribute_VERSION_1;
partyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
partyAttribute.uValue.m_nVal = PARM_VAL_Party_Mode_FULL;

partyInfo.m_nVersion = TSPartyInfo_VERSION_1;
partyInfo.m_nHandle = voiceHandle;
partyInfo.m_nAttrCount = 1;
partyInfo.m_pAttrs = &partyAttribute;
partyInfo.m_nPartyID = 0;

partyList.m_nVersion = TSPartyList_VERSION_1;
partyList.m_nPartyCount = 1;
partyList.m_ppartyInfo = &partyInfo;

/* set up the user context */
g_partyUserContext.threadID = GetCurrentThreadId();
g_partyUserContext.index = 2;/* user's own number */
g_partyUserContext.byTimeslot = 0;
```

```
if (cnf_AddParty(g_confHandle,
             &partyList,
             &g_partyUserContext,
             EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_AddParty on conference handle %d", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

/* set board attribute */
getPartyAttribute.m_nVersion = TSAttribute_VERSION_1;
getPartyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
getPartyAttributeList.m_nAttrCount = 1;
getAttributesList.m_nVersion = TSAttribute_VERSION_1;
getAttributesList.m_pAttrs = &getPartyAttribute;

if(cnf_GetPartyAttributes(g_confHandle,
          g_PartyID,
          &getAttributesList,
          NULL,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_GetPartyAttributes on party ID %d", g_PartyID);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
}

/* wait for the event */
sr_waitevt(-1);

/* Remove the party

   cnf_RemoveParty(…);
*/

/* wait for the event */
sr_waitevt(-1);

if(cnf_DeleteConference(g_boardHandle,
          g_confHandle,
          &g_strUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);
```

```
     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     bool  byTimeslot;

     strUserContext           *pstrUserContext;
     TSPartyList               *pPartyListReply;
     TSAttributesList          *pAttributesListReply
     void                      *pUserContext;

     eventType = sr_getevttype();               /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
           threadID =      pstrUserContext->threadID;
           index =      pstrUserContext->index;
           byTimeslot = pstrUserContext->byTimeslot;
     }
     switch(eventType)
     {
          case CNFEV_PARTYATTR:
          /* retrieve the reply */
          pAttributesListReply = (TSAttributesList*)(pEventData);

          for(int count = 0;
              count < pAttributesListReply->m_nAttrCount;
              count++)
          {
               printf("Party Attribute Type: 0x%x\n"\
                         pAttributesListReply->m_pAttrs->m_nAttrType;
               printf("Party Attribute Value: %d\n"\
                         pAttributesListReply->m_pAttrs->uValue.m_nVal);

               pAttributesListReply->m_pAttrs++;
          }
          break;

          case CNFEV_GETPARTYATTRFAILED:

          printf("Got CNFEV_GETPARTYATTRFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

          break;

          case CNFEV_PARTYADDED:
```

```
            printf("Got CNFEV_PARTYADDED event.\n");
            if (!byTimeslot)
            {
                 /* retrieve the reply */
                 pPartyListReply = (TSpartyList*)(pEventData);

                 printf("Party ID: %d\n", pPartyListReply->m_ppartyInfo->m_nPartyID);
                 g_PartyID = pPartyListReply->m_ppartyInfo->m_nPartyID;
            }
            break;

            case CNFEV_ADDPARTYFAILED:

            printf("Got CNFEV_ADDPARTYFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
            break;

            :
            :
            default:
            break;
       }

       return (0);
}
```

See Also

- **cnf_SetPartyAttributes( )**

cnf_GetPartyList( )

Name: int cnf_GetPartyList(a_nConfHandle, a_partyList, a_pUserContext, a_nMode)

Inputs:	int a_nConfHandle	• conference device handle
	TSPartyList *a_partyList	• pointer to list of parties in a conference
	void *a_pUserContext	• user-supplied pointer used in asynchronous mode
	int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetPartyList( )** function returns the list of parties in a conference.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_partyList	specifies the list of parties in a conference. For more information, see the data structure description for TSPartyList, on page 135. In asynchronous mode, specify NULL. The list of parties will be part of the event data.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

In synchronous mode, the application is responsible for allocating sufficient memory for the TSPartyList structure.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_PARTYLIST
Get party list completed successfully

CNFEV_GETPARTYLISTFAILED
Get party list failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
char    g_confName[20] = {0};
long evt_handler(unsigned long a_evtHandle);

int main()
{
     /* The existing conference name, passed in by another process */
     char *pAttachConfName = GetConferenceName()

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;/* user's own number */
```

```
    if(cnf_AttachConference(g_boardHandle,
              pAttachConfName,
              NULL,
              &g_strUserContext,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_AttachConference on board handle %d\n", g_boardHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if (cnf_GetPartyList(g_confHandle,
              NULL,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_ GetPartyList on conference handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if(cnf_DeleteConference(g_boardHandle,
              g_confHandle,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
    {
         printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         return(-1);
    }

    return (0);
}
```

```
long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     int   eventDevice;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     TSPartyList              *pPartyListReply;
     void                     *pUserContext;

     eventType = sr_getevttype();                   /* check the event type */
     pEventData = sr_getevtdatap();                 /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
          threadID =   pstrUserContext->threadID;
          index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_PARTYLIST:

          printf("Got CNFEV_PARTYLIST event.\n");

          /* retrieve the reply */
          pPartyListReply = (TSpartyList*)(pEventData);

          printf("Number of active talkers: %d\n", pPartyListReply->m_nPartyCount);

          int numofActiveTalkers = pPartyListReply->m_nPartyCount;
          for (int count = 0; count< numofActiveTalkers; count++)
          {
               printf("Party ID: %d\n", pPartyListReply->m_pPartyInfo->m_nPartyID);

               pPartyListReply->m_pPartyInfo++;
          }

          break;

          case CNFEV_GETPARTYLISTFAILED:
          printf("Got CNFEV_GETPARTYLISTFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
          break;

          case CNFEV_CONFATTACHED:

          printf("Got CNFEV_CONFATTCHED event.\n");

          /* retrieve the conference reply */
          pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

          printf("The Attached Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
          printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

          printf("Index from user context = %d\n", index);

          /* set conference handle */
          g_confHandle = pTSCreateConfReply->m_nConfHandle);
```

```
            /* set conference name */
            strcpy(g_confName, pTSCreateConfReply->m_szConfName);

            break;

            case CNFEV_ATTACHCONFFAILED:

            printf("Got CNFEV_ATTACHCONFFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            break;

                 :
                 :

            default:
            break;
      }
      return (0);
}
```

See Also

- **cnf_GetActiveTalkers( )**

cnf_GetResourceCount( )

Name: int cnf_GetResourceCount(a_nBrdHandle, a_pRes, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
CNF_RES * a_pRes	• pointer to structure containing resources available
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetResourceCount( )** function returns the number of parties or resources available on the specified board. This function returns the number of free parties and the maximum number of parties supported.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pRes	points to a structure containing the number of parties or resources available. For more information, see the data structure description for CNF_RES, on page 121. In asynchronous mode, specify NULL.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_RESCOUNT
Get resource count completed successfully

CNFEV_GETRESCOUNTFAILED
Get resource count failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
long    evt_handler(unsigned long a_evtHandle);
int     lPartyGroup;

int main()
{
     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;/* user's own number */
```

```
     if ((cnf_GetResourceCount (g_boardHandle,
               NULL,
               &g_strUserContext,
               EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_GetResourceCount on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
}

     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return(-1);
     }

     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     int   eventDevice;

     strUserContext           *pstrUserContext;
     CNF_RES                  *pResourceCountReply;
     void                     *pUserContext;

     eventType = sr_getevttype();                  /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
          threadID =   pstrUserContext->threadID;
          index =      pstrUserContext->index;
     }

     switch(eventType)
     {
        case CNFEV_RESCOUNT:

        printf("Got CNFEV_RESCOUNT event.\n");
        pResourceCountReply = (CNF_RES*)(pEventData);
        printf("Total MaxParties = %d\n", pResourceCountReply->m_nBrdMaxParties);
        printf("Total FreeParties = %d\n", pResourceCountReply->m_nBrdFreeParties);
        printf("Total MaxConferences = %d\n", pResourceCountReply->m_nBrdMaxConferences);
        printf("Total FreeConferences = %d\n", pResourceCountReply->m_nBrdFreeConferences);
        lPartyGroup = 0;
        while (lPartyGroup < pResourceCountReply->m_nGroupCount)
        {
             printf("Party Group %d available resources:\n",
                       pResourceCountReply->m_pGroups[lPartyGroup].m_nPartyGroup);
             printf("MaxParties = %d\n",
```

```
                    pResourceCountReply>m_pGroups[lPartyGroup].m_nGrpMaxParties);
               printf("FreeParties = %d\n",
                    pResourceCountReply->m_pGroups[lPartyGroup].m_nGrpFreeParties);
               printf("MaxConferences = %d\n",
                    pResourceCountReply->m_pGroups[lPartyGroup].m_nGrpMaxConferences);
               printf("FreeConferences = %d\n",
                    pResourceCountReply->m_pGroups[lPartyGroup].m_nGrpFreeConferences);
               lPartyGroup++;

            }

            break;

            case CNFEV_GETRESCOUNTFAILED:
            printf("Got CNFEV_GETRESCOUNTFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",
                 ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            break;
                 :
                 :

            default:
            break;
         }
         return (0);
}
```

See Also

None

cnf_GetVolumeControl( )

Name: int cnf_GetVolumeControl(a_nBrdHandle, a_pVol, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
CNF_VOL * a_pVol	• pointer to structure for volume control
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_GetVolumeControl( )** function returns information on the DTMF digits used to control the volume of a conference on a physical board.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pVol	points to a structure defining DTMF digits used to control the volume of a conference. For more information, see the data structure description for CNF_VOL, on page 122.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_VOLCONTROL
Get volume control completed successfully

CNFEV_GETVOLCONTROLFAILED
Get volume control failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
long evt_handler(unsigned long a_evtHandle);

int main()
{
     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;/* user's own number */

     if ((cnf_GetVolumeControl(g_boardHandle,
               NULL,
               &g_strUserContext,
               EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_GetVolumeControl on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }
```

```
     /* wait for the event */
     sr_waitevt(-1);

     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return(-1);
     }
     return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     int   eventDevice;

     strUserContext        *pstrUserContext;
     CNF_VOL               *pVolumeControlReply
     void                  *pUserContext;

     eventType = sr_getevttype();                  /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
           threadID =      pstrUserContext->threadID;
           index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_VOLCONTROL:

          printf("Got CNFEV_VOLCONTROL event.\n");

          pVolumeControlReply = (CNF_VOL*)(pEventData);

          printf("Up digit: %d\n", pVolumeControlReply->m_volUpDigit);
          printf("Down digit: %d\n", pVolumeControlReply->m_volDownDigit);

          break;

          case CNFEV_GETVOLCONTROLFAILED:
          printf("Got CNFEV_GETVOLCONTROLFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
          break;
                  :
                  :

          default:
          break;
     }
     return (0);
}
```

See Also

- cnf_SetVolumeControl( )

cnf_Open( )

Name: int cnf_Open(a_pnBrdHandle, a_szName, a_oFlags)

Inputs:

int *a_pnBrdHandle	• pointer to the physical board device handle
char* a_szName	• pointer to the physical board device name
int a_oFlags	• flag reserved for future use

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Device Management

Mode: Synchronous

Platform: DM3

Description

The **cnf_Open( )** function opens a physical board device and returns a handle to that board. The device naming convention for the physical board is **brdBn**, where **n** is the board number starting at 1. To create a conference on a particular board, you must open the physical board device and pass the returned handle as an argument to **cnf_CreateConference( )** or **cnf_AttachConference( )**.

Parameters	Description
a_pnBrdHandle	specifies a pointer to the valid physical board device handle returned by **cnf_Open( )**
a_szName	specifies a pointer to the name of the physical board device in the format **brdBn** (for example brdB1)
a_oFlags	flag reserved for future use. Specify 0.

Cautions

Since the same physical board device can be opened in multiple processes, the other process can delete conferences running on the physical board. The application is responsible for synchronizing access to the same physical board from multiple processes.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, “Error Codes”.

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int g_BoardHandle = -1;
void main()
{
     if ((cnf_Open(&g_BoardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open board\n");
     }
     else
     {
          printf("Got board handle: %d\n", g_BoardHandle);

     /* Perform other processing... */

     if ((cnf_Close(g_BoardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close &s\n", ATDV_NAMEP(g_BoardHandle));
          printf("error code = 0x%x, error message = %s\n",
                    ATDV_LASTERR(g_BoardHandle), ATDV_ERRMSGP(g_BoardHandle));
     }
}
```

See Also

- cnf_Close( )
- cnf_CreateConference( )

cnf_RemoveParty( )

Name: int cnf_RemoveParty(a_nConfHandle, a_partyList, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle	• conference device handle
TSPartyList *a_partyList	• pointer to party specific information
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Conference Management

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_RemoveParty( )** function removes a party (conferee) from a conference.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**.
a_partyList	specifies a pointer to party specific information. The party to be removed is identified by m_pPartyInfo[x].m_nPartyID. For more information, see the data structure description for TSPartyList, on page 135.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_PARTYREMOVED
Party removed successfully

CNFEV_REMOVEPARTYFAILED
Remove party operation failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confUserContext;
strUserContext   g_partyUserContext;

int     g_boardHandle = -1;
int     g_confHandle = -1;
int     g_partyID = -1;
char    g_confName[20] = {0};
long evt_handler(unsigned long a_evtHandle);

int main()
{
     int g_boardHandle;
     TSConferenceInfo conferenceInfo;
     TSConferenceAttribute conferenceAttribute;

     int voiceHandle = 0;
     TSPartyAttribute partyAttribute;
     TSPartyInfo partyInfo;
     TSPartyList partyList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }
```

```
if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
{
     printf("Failed to open brdB1\n");
     return (-1);
}

conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
conferenceAttribute.uValue.m_nVal = 1;

conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_confUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

voiceHandle = dx_open("dxxxB1C1", 0);
if (voiceHandle < 0)
{
     printf("Failed to open dxxxB1C1\n");

     /* ProcessError( ); */
     return (-1);
}

partyAttribute.m_nVersion = TSAttribute_VERSION_1;
partyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
partyAttribute.uValue.m_nVal = PARM_VAL_Party_Mode_FULL;

partyInfo.m_nVersion = TSPartyInfo_VERSION_1;
partyInfo.m_nHandle = voiceHandle;
partyInfo.m_nAttrCount = 1;
partyInfo.m_pAttrs = &partyAttribute;
partyInfo.m_nPartyID = 0;

partyList.m_nVersion = TSPartyList_VERSION_1;
partyList.m_nPartyCount = 1;
partyList.m_ppartyInfo = &partyInfo;

/* set up the user context */
g_partyUserContext.threadID = GetCurrentThreadId();
g_partyUserContext.index = 2;
g_partyUserContext.byTimeslot = 0;
```

```
    if (cnf_AddParty(g_confHandle,
         &partyList,
         &g_partyUserContext,
         EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_AddParty on conference handle %d",\   g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

         /* ProcessError() */
         return (-1);

    }

    /* wait for the event */
    sr_waitevt(-1);

    partyInfo.m_nPartyID = g_PartyID;

    if (cnf_RemoveParty(g_confHandle,
              &partyList,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_RemoveParty on conference handle %d", g_confHandle);
         printf("Error code: %d, Error Message: %s",\ ATDV_NAMEP(g_confHandle),
                   ATDV_ERRMSGP(g_confHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if(cnf_DeleteConference(g_boardHandle,
              g_confHandle,
              NULL,
              EV_ASYNC) == CNF_FAILURE)
    {
         printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

         /* ProcessError() */
         return (-1);
    }

    /* wait for the event */
    sr_waitevt(-1);

    if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
    {
         printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
         printf("Error code = 0x%x, Error message = %s\n",\
                   ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
         return (-1);
    }

    return (0);
}
```

```
long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;
     bool  byTimeslot;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     void                     *pUserContext;

     eventType = sr_getevttype();                    /* check the event type */
     pEventData = sr_getevtdatap();                  /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);
     if (pstrUserContext != NULL)
     {
          /* Retrieve user context */
          threadID =   pstrUserContext->threadID;
          index =      pstrUserContext->index;
          byTimeslot = pstrUserContext->byTimeslot;
     }

     switch(eventType)
     {
          case CNFEV_PARTYREMOVED:

          printf("Got CNFEV_PARTYREMOVED event.\n");
          break;

          case CNFEV_REMOVEPARTYFAILED:

          printf("Got CNFEV_REMOVEPARTYFAILED event.\n");
          printf("Error code: %d, Error Message: %s", ATDV_NAMEP(g_confHandle),\
                    ATDV_ERRMSGP(g_confHandle));

          break;

          :
          :
          default:
          break;
     }

     return (0);
}
```

See Also

- **cnf_AddParty( )**

cnf_SetBoardAttributes( )

Name: int cnf_SetBoardAttributes(a_nBrdHandle, a_pAttributes, a_pUserContext, a_nMode)

Inputs:

int a_nBrdHandle	• physical board device handle
TSAttributesList * a_pAttributes	• pointer to list of attributes
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_SetBoardAttributes( )** function sets the values of one or more physical board attributes.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pAttributes	specifies a pointer to the list of attributes for a physical board device. For more information, see the data structure description for TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

The function returns success only if all the specified attributes can be set for the board. If not, the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_BRDATTRSET
Set board attributes completed successfully

CNFEV_SETBRDATTRFAILED
Set board attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

int     g_boardHandle = -1;
long evt_handler(unsigned long a_evtHandle);

main()
{
     TSBoardAttribute      setBoardAttribute;
     TSAttributesList      setAttributesList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&BoardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
     }
     else
     {
          printf("Got board Handle: %d\n", BoardHandle);
     }

     setBoardAttribute.m_nVersion = TSAttribute_VERSION_1;
     setBoardAttribute.m_nAttrType = BOARD_Parm_Act_Talk_En;
     setBoardAttribute.uValue.m_nVal = 1;

     setAttributesList.m_nVersion = TSAttributesList_VERSION_1;
     setAttributesList.m_nAttrCount = 1;
     setAttributesList.m_pAttrs = &setBoardAttribute;

     if ((cnf_SetBoardAttributes(BoardHandle,
               &setAttributesList,
               NULL, EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_SetBoardAttributes on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          /* ProcessError() */
          return (-1);
     }

     /* wait for the event */
     sr_waitevt(-1);
```

```
     if( (cnf_Close(BoardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(BoardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(BoardHandle), ATDV_ERRMSGP(BoardHandle));
     }

}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext            *pstrUserContext;
     TSCreateConferenceReply   *pTSCreateConfReply;
     void                      *pUserContext;

     eventType = sr_getevttype();                  /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);

     if (pstrUserContext != NULL)
     {
           /* Retrieve user context */
           threadID =   pstrUserContext->threadID;
           index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_BRDATTRSET:

          printf("Got CNFEV_BRDATTRSET event.\n");
          break;

          case CNFEV_SETBRDATTRFAILED:
          printf("Got CNFEV_SETBRDATTRFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          break;

               :
               :
          default:
          break;
     }
     return (0);
}
```

See Also

- **cnf_GetBoardAttributes( )**

cnf_SetConferenceAttributes( )

Name: int cnf_SetConferenceAttributes(a_nConfHandle, a_pAttributes, a_pUserContext, a_nMode)

Inputs:	int a_nConfHandle	• conference device handle
	TSAttributesList *a_pAttributes	• pointer to list of attributes
	void *a_pUserContext	• user-supplied pointer used in asynchronous mode
	int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_SetConferenceAttributes( )** function sets the values of one or more conference attributes.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_pAttributes	specifies a pointer to the list of attributes for the conference device. For more information, see the data structure description for TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

The function returns success only if all the specified attributes can be set for the conference. If not, the function returns an error.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_CNFATTRSET
Set conference attributes completed successfully

CNFEV_SETCNFATTRFAILED
Set conference attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
int     g_confHandle =  -1;
char    g_confName[20];
int main()
{
     TSConferenceInfo           conferenceInfo;
     TSConferenceAttribute      conferenceAttribute;

     TSConferenceAttribute      setConfAttribute;
     TSAttributesList           setAttributesList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
     conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
     conferenceAttribute.uValue.m_nVal = 1;
```

```
conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_strUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

setAttribute.m_nVersion = TSAttribute_VERSION_1;
setConfAttribute.m_nAttrType = CONF_Parm_DTMF_Mask;
setConfAttribute.uValue.m_nVal = CNF_DIG0 | CNF_DIG_ADD_MASK;

setAttributesList.m_nVersion = TSAttributesList_VERSION_1;
setAttributesList.m_nAttrCount = 1;
setAttributesList.m_pAttrs = &setConfAttribute;

if ((cnf_SetConferenceAttributes(g_confHandle,
          &setAttributesList,
          NULL,
          EV_ASYNC)) == CNF_FAILURE)
{
     printf("Failed to issue cnf_ SetConferenceAttributes \
                on conference handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
                ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);

if(cnf_DeleteConference(g_boardHandle,
          g_confHandle,
          &g_strUserContext,
          EV_ASYNC) == CNF_FAILURE)
{
     printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return (-1);
}

/* wait for the event */
sr_waitevt(-1);
```

```
     if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
     {
          printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

          return (-1);
     }

     return (0);

}

long evt_handler(unsigned long a_evtHandle)
{
     PVOID pEventData;
     int   eventType;
     int   threadID;
     int   index;

     strUserContext           *pstrUserContext;
     TSCreateConferenceReply  *pTSCreateConfReply;
     void                     *pUserContext;

     eventType = sr_getevttype();                  /* check the event type */
     pEventData = sr_getevtdatap();                /* get the event data p */
     pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
     pstrUserContext = (strUserContext*)(pUserContext);

if (pstrUserContext != NULL)
     {
          /* Retrieve user context */
          threadID =   pstrUserContext->threadID;
          index =      pstrUserContext->index;
     }

     switch(eventType)
     {
          case CNFEV_CONFATTRSET:

          printf("Got CNFEV_CONFATTRSET event.\n");
          break;

          case CNFEV_SETCONFATTRFAILED:
          printf("Got CNFEV_SETCONFATTRFAILED event.\n");
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

          break;
          case CNFEV_CONFCREATED:

          printf("Got CNFEV_CONFCREATED event. \n");
          /* retrieve the conference reply */
          pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

          printf("The Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
          printf("The Attached Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

          printf("Index from user context = %d\n", index);

          /* set conference handle */
          g_confHandle = pTSCreateConfReply->m_nConfHandle);

          /* set conference name */
          strcpy(g_confName, pTSCreateConfReply->m_szConfName);

          break;
```

```
            case CNFEV_CREATECONFFAILED:

            printf("Got CNFEV_CREATECONFFAILED event.\n");

            /* retrieve the conference reply */
            pTSCreateConfReply = (TSCreateConferenceReply*)(pEventData);

            printf("The Conference Name: %s\n", pTSCreateConfReply->m_szConfName);
            printf("The Conference Handle: %d\n", pTSCreateConfReply->m_nConfHandle);

            printf("Index from user context = %d\n", index);

            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

            break;
                 :
                 :
            default:
            break;
        }
        return (0);
}
```

See Also

- **cnf_GetConferenceAttributes( )**

cnf_SetPartyAttributes( )

Name: int cnf_SetPartyAttributes(a_nConfHandle, a_partyID, a_pAttributes, a_pUserContext, a_nMode)

Inputs:

int a_nConfHandle	• conference device handle
int a_partyID	• party ID
TSAttributesList * a_pAttributes	• pointer to list of attributes
void *a_pUserContext	• user-supplied pointer used in asynchronous mode
int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_SetPartyAttributes( )** function sets the values of one or more party attributes.

Parameters	Description
a_nConfHandle	specifies the valid conference device handle returned by either **cnf_CreateConference( )** or **cnf_AttachConference( )**
a_partyID	specifies the party identifier returned by **cnf_AddParty( )**
a_pAttributes	specifies the list of attributes for the party. For more information, see the data structure description for TSAttributesList, on page 127.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

- The function returns success only if all the specified attributes can be set for the party. If not, the function returns an error.
- In asynchronous mode, the party ID is not returned with the event. To distinguish between multiple **cnf_SetPartyAttributes( )** calls on the same conference, use **a_pUserContext** to

pass in the party ID. For more information user context, see the *Conferencing API Programming Guide*.

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_PARTYATTRSET
Set party attributes completed successfully

CNFEV_SETPARTYATTRFAILED
Set party attributes failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "dxxxlib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_confUserContext;
strUserContext   g_partyUserContext;

int     g_boardHandle = -1;
int     g_confHandle = 0;
int     g_partyID = 0;
char    g_confName[20];
int main()
{
     int voiceHandle = 0;

     TSconferenceInfo           conferenceInfo;
     TSconferenceAttribute      conferenceAttribute;

     TSpartyAttribute           partyAttribute;
     TSpartyInfo                partyInfo;
     TSpartyList                partyList;

     TSPartyAttribute      setPartyAttribute[2];
     TSAttributesList     setPartyAttributeList;

     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);
```

```
if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
{
     printf("sr_enbhdlr failed to assign the handler\n");
     return (-1);
}

if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == -1)
{
     printf("Failed to open brdB1\n");
     return (CNF_FAILURE);
}

conferenceAttribute.m_nVersion = TSAttribute_VERSION_1;
conferenceAttribute.m_nAttrType = CONF_Parm_ToneClamping;
conferenceAttribute.uValue.m_nVal = 1;

conferenceInfo.m_nVersion = TSConferenceInfo_VERSION_1;
conferenceInfo.m_nPartyGroup = 0;
conferenceInfo.m_nAttrCount = 1;
conferenceInfo.m_pAttrs = &conferenceAttribute;

/* set up the user context */
g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.index = 1;/* user's own number */

if(cnf_CreateConference(g_boardHandle,
          &conferenceInfo,
          NULL,
          &g_confUserContext,
          EV_ASYNC) == -1)
{
     printf("Failed to issue cnf_CreateConference on board handle %d\n", g_boardHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

     /* ProcessError() */
     return CNF_FAILURE;
}

sr_waitevt(-1);

voiceHandle = dx_open("dxxxB1C1", 0);
if (voiceHandle < 0)
{
     printf("Failed to open dxxxB1C1\n");

/* ProcessError( ); */
return (CNF_FAILURE);
}

partyAttribute.m_nVersion = TSAttribute_VERSION_1;
partyAttribute.m_nAttrType = PARTY_Parm_Party_Mode;
partyAttribute.uValue.m_nVal = PARM_VAL_Party_Mode_FULL;

partyInfo.m_nVersion = TSPartyInfo_VERSION_1;
partyInfo.m_nHandle = voiceHandle;
partyInfo.m_nAttrCount = 1;
partyInfo.m_pAttrs = &partyAttribute;
partyInfo.m_nPartyID = 0;

partyList.m_nVersion = TSPartyList_VERSION_1;
partyList.m_nPartyCount = 1;
partyList.m_ppartyInfo = &partyInfo;
```

```
/* set up the user context */
g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.index = 2;/* user's own number */
g_strUserContext.byTimeslot = 0;

if (cnf_AddParty(g_confHandle,
             &partyList,
             &g_partyUserContext,
             EV_ASYNC) == -1)
{
     printf("Failed to issue cnf_AddParty on conference handle %d", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
     return (CNF_FAILURE);
}

sr_waitevt(-1);

/* set board attribute */
setPartyAttribute[0].m_nVersion = TSAttribute_VERSION_1;
setPartyAttribute[0].m_nAttrType = PARTY_Parm_Coach_En;
setPartyAttribute[0].uValue.m_nVal = 1;
setPartyAttribute[1].m_nVersion = TSAttribute_VERSION_1;
setPartyAttribute[1].m_nAttrType = PARTY_Parm_Tariff_En;
setPartyAttribute[1].uValue.m_nVal = 1;

setAttributesList.m_nVersion = TSAttributesList_VERSION_1;
setAttributesList.m_nAttrCount = 2;
setAttributesList.m_pAttrs = setPartyAttribute;

g_strUserContext.threadID = GetCurrentThreadId();
g_strUserContext.m_nIndex = 3;/* user's own number */

if(cnf_SetPartyAttributes(g_confHandle,
          g_PartyID,
          &setAttributesList,
          NULL,
          EV_ASYNC) == -1)
{
     printf("Failed to issue cnf_SetPartyAttributes on party ID %d", g_PartyID);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));

     /* ProcessError() */
}

sr_waitevt(-1);

/* Remove the party

 cnf_RemoveParty(…);
*/

sr_waitevt(-1);

if(cnf_DeleteConference(g_boardHandle,
          g_confHandle,
          &g_strUserContext,
          EV_ASYNC) == -1)
{
     printf("Failed to issue cnf_DeleteConference on conf handle %d\n", g_confHandle);
     printf("Error code = 0x%x, Error message = %s\n",\
               ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
```

```
            /* ProcessError() */
            return CNF_FAILURE;
        }

        sr_waitevt(-1);

        if( (cnf_Close(g_boardHandle)) == -1)
        {
            printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
            printf("Error code = 0x%x, Error message = %s\n",\
                      ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

            return (CNF_FAILURE);
        }

        return (CNF_SUCCESS);
    }

    long evt_handler(unsigned long a_evtHandle)
    {
        PVOID pEventData;
        int   eventType;
        int   threadID;
        int   index;
        int   nMaximum;
        int   eventDevice;
        int   nDeviceHandle;
        char *pDeviceName;
        bool  byTimeslot;
        strUserContext          *pstrUserContext;
        TSPartyList             *pPartyListReply;
        void                    *pUserContext;

        eventType = sr_getevttype();               /* check the event type */
        pEventData = sr_getevtdatap();                /* get the event data p */
        pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
        pstrUserContext = (strUserContext*)(pUserContext);
        if (pstrUserContext != NULL)
        {
              /* Retrieve user context */
              threadID =      pstrUserContext->threadID;
              index =      pstrUserContext->index;
              byTimeslot = pstrUserContext->byTimeslot;
        }

        switch(eventType)
        {
            case CNFEV_PARTYATTRSET:

            printf("Got CNFEV_PARTYATTRSET event.\n");
            break;

            case CNFEV_SETPARTYATTRFAILED:

            printf("Got CNFEV_SETPARTYATTRFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
            ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
            break;

            case CNFEV_PARTYADDED:

            printf("Got CNFEV_PARTYADDED event.\n");
            if (!byTimeslot)
            {
                /* retrieve the reply */
                pPartyListReply = (TSpartyList*)(pEventData);
```

```
                printf("Party ID: %d\n", pPartyListReply->m_ppartyInfo->m_nPartyID);
                g_PartyID = pPartyListReply->m_ppartyInfo->m_nPartyID;
            }

            break;

            case CNFEV_ADDPARTYFAILED:

            printf("Got CNFEV_ADDPARTYFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
            ATDV_LASTERR(g_confHandle), ATDV_ERRMSGP(g_confHandle));
            break;

            :
            :
            default:
            break;
        }

        return (0);
}
```

See Also

- **cnf_GetPartyAttributes( )**

cnf_SetVolumeControl( )

Name: int cnf_SetVolumeControl(a_nBrdHandle, a_pVol, a_pUserContext, a_nMode)

Inputs:	int a_nBrdHandle	• physical board device handle
	CNF_VOL * a_pVol	• pointer to structure
	void *a_pUserContext	• user-supplied pointer used in asynchronous mode
	int a_nMode	• mode of operation

Returns: CNF_SUCCESS for success
CNF_FAILURE for failure

Includes: srllib.h
cnflib.h

Category: Configuration

Mode: Synchronous/Asynchronous

Platform: DM3

Description

The **cnf_SetVolumeControl( )** function defines DMTF digits used to control the volume of a conference. Digits can be defined to increase, decrease, or reset the volume.

Parameters	Description
a_nBrdHandle	specifies the valid physical board device handle returned by **cnf_Open( )**
a_pVol	points to the structure defining DMTF digits used to control the volume of a conference. For more information, see the data structure description for CNF_VOL, on page 122.
a_pUserContext	in asynchronous mode, specifies a user-supplied pointer that can be retrieved using **sr_getUserContext( )** when the completion event is received. For more information on this function, see the *Standard Runtime Library API Library Reference*.
a_nMode	specifies the mode of operation: • EV_ASYNC – asynchronous • EV_SYNC – synchronous

Cautions

None

Errors

If this function returns CNF_FAILURE, use the Standard Runtime Library (SRL) Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message.

For a list of error codes, see Chapter 5, "Error Codes".

Events

CNFEV_VOLCONTROLSET
Set volume control completed successfully

CNFEV_SETVOLCONTROLFAILED
Set volume control failed

Example

```
#include <stdio.h>
#include "srllib.h"
#include "cnflib.h"

/* User defined structure */
typedef struct {
     int     threadID;
     int     index;
     bool    byTimeslot;
     int     timeslot;
} strUserContext;

strUserContext   g_strUserContext;

int     g_boardHandle = -1;
long evt_handler(unsigned long a_evtHandle);

int main()
{
     int srlMode = SR_POLLMODE;
     sr_setparm(SRL_DEVICE, SR_MODEID, &srlMode);

     if (sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, evt_handler) == -1)
     {
          printf("sr_enbhdlr failed to assign the handler\n");
          return (-1);
     }

     if ((cnf_Open(&g_boardHandle, "brdB1", 0)) == CNF_FAILURE)
     {
          printf("Failed to open brdB1\n");
          return (-1);
     }

     setVolControl.m_nVersion = CNF_VOL_VERSION_1;
     setVolControl.m_volEnable = 1;
     setVolControl.m_volUpDigit = CNF_DIG0;
     setVolControl.m_volDownDigit = CNF_DIG1;
     setVolControl.m_volResetDigit = CNF_DIGSTAR;

     /* set up the user context */
     g_strUserContext.threadID = GetCurrentThreadId();
     g_strUserContext.index = 1;/* user's own number */

     if ((cnf_SetVolumeControl(g_boardHandle,
               &setVolControl,
               &g_strUserContext,
               EV_ASYNC)) == CNF_FAILURE)
     {
          printf("Failed to issue cnf_ SetVolumeControl on board handle %d\n", g_boardHandle);
          printf("Error code = 0x%x, Error message = %s\n",\
                    ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
```

```
            /* ProcessError() */
            return (-1);
      }

      /* wait for the event */
      sr_waitevt(-1);

      if( (cnf_Close(g_boardHandle)) == CNF_FAILURE)
      {
            printf("Failed to close %s\n", ATDV_NAMEP(g_boardHandle));
            printf("Error code = 0x%x, Error message = %s\n",\
                  ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));

            return(-1);
      }

      return (0);
}

long evt_handler(unsigned long a_evtHandle)
{
      PVOID pEventData;
      int   eventType;
      int   threadID = 0;
      int   index;
      int   eventDevice;

      strUserContext          *pstrUserContext;
      void                     *pUserContext;

      eventType = sr_getevttype();                  /* check the event type */
      pEventData = sr_getevtdatap();                /* get the event data p */
      pUserContext = sr_getUserContext(a_evtHandle); /* get user context */
      pstrUserContext = (strUserContext*)(pUserContext);
      if (pstrUserContext != NULL)
      {
            /* Retrieve user context */
            threadID =      pstrUserContext->threadID;
            index =      pstrUserContext->index;
      }

      switch(eventType)
      {
            case CNFEV_VOLCONTROLSET:

            printf("Got CNFEV_VOLCONTROLSET event.\n");
            break;

            case CNFEV_SETVOLCONTROLFAILED:
            printf("Got CNFEV_SETVOLCONTROLFAILED event.\n");
            printf("Error code = 0x%x, Error message = %s\n",\
                  ATDV_LASTERR(g_boardHandle), ATDV_ERRMSGP(g_boardHandle));
            break;
                  :
                  :

            default:
            break;
      }
      return (0);
}
```

See Also

- cnf_GetVolumeControl( )

Events 3

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

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

Events in the conferencing library can be categorized as follows:

- termination events, which are produced when a function running in asynchronous mode terminates.
- unsolicited events, which are not generated in response to the completion of a function. Rather, they are generated in response to a condition of a given function.

The conferencing library provides a pair of termination events for each function, to indicate successful completion or failure. To collect termination event codes, use Standard Runtime Library (SRL) functions such as **sr_waitevt( )** and **sr_enbhdlr( )** depending on the programming model in use. For more information, see the Standard Runtime Library documentation.

The following events may be returned by the conferencing software:

CNFEV_ACTTALKERS
Termination event for **cnf_GetActiveTalkers( )**. Get active talkers completed successfully.

CNFEV_ADDPARTYFAILED
Termination event for **cnf_AddParty( )** or **cnf_AddPartyByTimeslot( )**. Failed to add parties.

CNFEV_ALLCONFDELETED
Termination event for **cnf_DeleteAllConferences( )**. All conferences deleted successfully.

CNFEV_ATTACHCONFFAILED
Termination event for **cnf_AttachConference( )**. Open conference operation failed.

CNFEV_BRDATTR
Termination event for **cnf_GetBoardAttributes( )**. Get board attributes operation completed successfully.

CNFEV_BRDATTRSET
Termination event for **cnf_SetBoardAttributes( )**. Set board attributes operation completed successfully.

CNFEV_BRIDGECREATED
Termination event for **cnf_CreateBridge( )**. Two conferences bridged successfully.

CNFEV_BRIDGEDELETED
Termination event for **cnf_DeleteBridge( )**. Bridge deleted successfully.

CNFEV_CONFATTACHED
Termination event for **cnf_AttachConference( )**. Conference opened successfully.

CNFEV_CONFATTR
Termination event for **cnf_GetConferenceAttributes( )**. Get conference attributes operation completed successfully.

CNFEV_CONFATTRSET
Termination event for **cnf_SetConferenceAttributes( )**. Set conference attributes operation completed successfully.

CNFEV_CONFCREATED
Termination event for **cnf_CreateConference( )**. Conference created successfully.

CNFEV_CONFDELETED
Termination event for **cnf_DeleteConference( )**. Conference deleted successfully.

CNFEV_CREATEBRIDGEFAILED
Termination event for **cnf_CreateBridge( )**. Bridging two conferences failed.

CNFEV_CREATECONFFAILED
Termination event for **cnf_CreateConference( )**. Create conference failed.

CNFEV_DELETEALLCONFFAILED
Termination event for **cnf_DeleteAllConferences( )**. Delete all conferences failed.

CNFEV_DELETEBRIDGEFAILED
Termination event for **cnf_DeleteBridge( )**. Delete bridge failed.

CNFEV_DELETECONFFAILED
Termination event for **cnf_DeleteConference( )**. Delete conference failed.

CNFEV_DISABLEEVTFAILED
Termination event for **cnf_DisableEvents( )**. Disable events for a board or conference failed.

CNFEV_DTMF
Unsolicited event that is sent on the conference handle when a conference detects DTMF digits. The digits are returned with the CNFEV_DTMF event. This event is enabled using **cnf_EnableEvents( )**.

CNFEV_ENABLEEVTFAILED
Termination event for **cnf_EnableEvents( )**. Enable events for a board or conference failed.

CNFEV_EVTDISABLED
Termination event for **cnf_DisableEvents( )**. Events for a board or conference disabled successfully.

CNFEV_EVTENABLED
Termination event for **cnf_EnableEvents( )**. Events for a board or conference enabled successfully.

CNFEV_GETACTTALKERSFAILED
Termination event for **cnf_GetActiveTalkers( )**. Get active talkers failed.

CNFEV_GETBRDATTRFAILED
Termination event for **cnf_GetBoardAttributes( )**. Get board attributes failed.

CNFEV_GETCONFATTRFAILED
Termination event for **cnf_GetConferenceAttributes( )**. Get conference attributes operation failed.

CNFEV_GETPARTYATTRFAILED
Termination event for **cnf_GetPartyAttributes( )**. Get party attributes operation failed.

CNFEV_GETPARTYLISTFAILED
Termination event for **cnf_GetPartyList( )**. Get party list failed.

CNFEV_GETRESCOUNTFAILED
Termination event for **cnf_GetResourceCount( )**. Get resource count completed successfully.

CNFEV_GETVOLCONTROLFAILED
Termination event for **cnf_GetVolumeControl( )**. Get volume control failed.

CNFEV_PARTYADDED
Termination event for **cnf_AddParty( )**. Party added successfully.

CNFEV_PARTYATTR
Termination event for **cnf_GetPartyAttributes( )**. Get party attributes completed successfully.

CNFEV_PARTYATTRSET
Termination event for **cnf_SetPartyAttributes( )**. Set party attributes completed successfully.

CNFEV_PARTYLIST
Termination event for **cnf_GetPartyList( )**. Get party list completed successfully.

CNFEV_PARTYREMOVED
Termination event for **cnf_RemoveParty( )**. Party removed successfully.

CNFEV_REMOVEPARTYFAILED
Termination event for **cnf_RemoveParty( )**. Remove party operation failed.

CNFEV_RESCOUNT
Termination event for **cnf_GetResourceCount( )**. Get resource count failed.

CNFEV_SETBRDATTRFAILED
Termination event for **cnf_SetBoardAttributes( )**. Set board attributes operation failed.

CNFEV_SETCONFATTRFAILED
Termination event for **cnf_SetConferenceAttributes( )**. Set conference attributes operation failed.

CNFEV_SETPARTYATTRFAILED
Termination event for **cnf_SetPartyAttributes( )**. Set party attributes operation failed.

CNFEV_SETVOLCONTROLFAILED
Termination event for **cnf_SetVolumeControl( )**. Set volume control failed.

CNFEV_STATUSADDPARTY
Unsolicited event that is sent to the conference device handle when a party is added using **cnf_AddParty( )**. This unsolicited event is different from the termination event as it can be sent to other processes attached to the same conference (using **cnf_AttachConference( )**) even though those processes did not initiate the add party operation. This unsolicited event allows all processes to be informed of changes in a given conference.

CNFEV_STATUSPARTYREMOVED
: Unsolicited event that is sent to the conference device handle when a party is removed using **cnf_RemoveParty( )**. This unsolicited event is different from the termination event as it can be sent to other processes attached to the same conference (using **cnf_AttachConference( )**) even though those processes did not initiate the remove party operation. This unsolicited event allows all processes to be informed of changes in a given conference.

CNFEV_VOLCONTROL
: Termination event for **cnf_GetVolumeControl( )**. Get volume control completed successfully.

CNFEV_VOLCONTROLSET
: Termination event for **cnf_SetVolumeControl( )**. Set volume control completed successfully.

Data Structures 4

This chapter provides an alphabetical reference to the data structures used by the conferencing library. The following data structures are discussed:

CNF_DIGINFO

```
typedef struct cnf_diginfo
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyID; /* Party Identifier */
     unsigned int m_nPartyGroup; /* Party Group # */
     unsigned char m_nDigits[MAX_CNFDIGS+1]; /* ASCIIZ Digits */
} CNF_DIGINFO;
```

Description

The CNF_DIGINFO data structure contains information about DTMF digits and the party that generated these digits. It is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is CNF_DIGINFO_VERSION_1.

m_nPartyID
: contains the party identification number

m_nPartyGroup
: contains the party group identifier

m_nDigits
: contains the digits entered by the specified party in the conference. The maximum number of digits is determined by the MAX_CNFDIGS define plus 1 for the NULL termination character. See *cnflib.h* for the MAX_CNFDIGS definition.

Example

None

CNF_RES

```
typedef struct
{
     unsigned int m_nVersion;
     unsigned short m_nBrdMaxParties;
     unsigned short m_nBrdFreeParties;
     unsigned short m_nBrdMaxConferences;
     unsigned short m_nBrdFreeConferences;
     unsigned int m_nGroupCount;
     TSPartyGroupRes *m_pGroups;
} CNF_RES;
```

Description

The CNF_RES data structure contains the total number of party groups, party resources, and conference resources available on a given physical board. This data structure is used by the **cnf_GetResourceCount( )** function. This data structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is CNF_RES_VERSION_1.

m_nBrdMaxParties
: specifies the maximum number of party resources available on the board

m_nBrdFreeParties
: specifies the number of free party resources available on the board

m_nBrdMaxConferences
: specifies the maximum number of conferences available on the board

m_nBrdFreeConferences
: specifies the number of free conferences available on the board

m_nGroupCount
: specifies the number of elements in the m_pGroups array

m_pGroups
: specifies an array of m_nGroupCount groups. For more information, see the data structure description for TSPartyGroupRes, on page 133.

Example

For an example of the CNF_RES data structure, see the Example section of the **cnf_GetResourceCount( )** function description.

CNF_VOL

```
typedef struct
{
     int m_nVersion;
     int m_volEnable;
     int m_volUpDigit;
     int m_volDownDigit;
     int m_volResetDigit;
} CNF_VOL;

typedef enum cnf_dig
{
     CNF_DIG0
     CNF_DIG1
     CNF_DIG2
     CNF_DIG3
     CNF_DIG4
     CNF_DIG5
     CNF_DIG6
     CNF_DIG7
     CNF_DIG8
     CNF_DIG9
     CNF_DIGSTAR
     CNF_DIGPOUND
     CNF_DIGA
     CNF_DIGB
     CNF_DIGC
     CNF_DIGD

     /*
      * These are used by CONF_Parm_DTMF_Mask to add, remove or set the DTMF
      * mask in cnf_SetConferenceAttributes().
      */
     CNF_DIG_ADD_MASK
     CNF_DIG_SUB_MASK
     CNF_DIG_SET_MASK
} CNF_DIG;
```

Description

The CNF_VOL data structure specifies whether or not volume control is active and contains definitions of DTMF digits used to control the volume of a conference. This data structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is CNF_VOL_VERSION_1.

m_volEnable
: enables or disables volume control on a physical board. Values are 1 to enable and 0 to disable.

m_volUpDigit
: specifies the digit used to increase the volume. This digit is specified using one of the values in the CNF_DIG enumeration. The volume increment is 2 dB.

m_volDownDigit
specifies the digit used to decrease the volume. This digit is specified using one of the values in the CNF_DIG enumeration. The volume decrement is 2 dB.

m_volResetDigit
specifies the digit used to reset the volume to its default level. This digit is specified using the CNF_DIG enumeration. The default volume and origin is 0 dB.

Example

For an example of the CNF_VOL data structure, see the Example section in **cnf_SetVolumeControl( )** function description.

TSAttribute

```
typedef struct SAttribute
{
     unsigned int m_nVersion;           /* Version of this structure */
     unsigned int m_nAttrType;
     /* One of: EPartyAttributes, EConferenceAttributes, EBoardAttributes */
     union
     {
          int m_nVal;
     } uValue;
} TSPartyAttribute, TSConferenceAttribute, TSBoardAttribute, TSAttribute;

typedef enum
{
     PARM_VAL_Party_Mode_NULL = 0, /* Null party means no timeslots assigned */
     PARM_VAL_Party_Mode_FULL      /* Full-duplex party */
     PARM_VAL_Party_Mode_RCV       /* Receive-only party (same as Monitor) */
     PARM_VAL_Party_Mode_XMIT      /* Transmit-only */
} EPARTY_Parm_Party_Mode;

typedef enum
{
     PARTY_Parm_Party_Mode         /* use EPARTY_Parm_Party_Mode */
     PARTY_Parm_Echo_Cancel_En     /* 1 = Enable, 0 = Disable */
     PARTY_Parm_Tariff_En          /* 1 = Enable, 0 = Disable */
     PARTY_Parm_AGC_En             /* 1 = Enable, 0 = Disable */
     PARTY_Parm_Broadcast_En       /* 1 = Enable, 0 = Disable */
     PARTY_Parm_Coach_En           /* 1 = Enable, 0 = Disable */
     PARTY_Parm_Pupil_En           /* 1 = Enable, 0 = Disable */
     PARTY_Parm_ToneClampingPty    /* 1 = Enable, 0 = Disable */
} EPartyAttributes;

typedef enum
{
     CONF_Parm_ToneClamping        /* 1 = Enable, 0 = Disable */
     CONF_Parm_DTMF_Mask           /* CNF_DIG values can be ORed to form the mask */
} EConferenceAttributes;

typedef enum
{
     BOARD_Parm_Act_Talk_En             /* 0 = Enable, 0xFFFF = Disable */
     BOARD_Parm_ToneClamping            /* 1 = Enable, 0 = Disable */
     BOARD_Parm_ActTalkerNotifyInterval /* Interval is x 100 ms. */
} EBoardAttributes;
```

Description

The TSAttribute data structure contains information about the attributes of a party, conference, or physical board. It has several aliases: TSPartyAttribute, TSConferenceAttribute, and TSBoardAttribute. This data structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSAttribute_VERSION_1.

m_nAttrType
specifies the type of attribute: party, conference, or board. Possible values are defined in these three enumerations:

- EPartyAttributes
- EConferenceAttributes
- EBoardAttributes

m_nVal
specifies the value of the parameter attribute indicated in EPartyAttributes, EConferenceAttributes or EBoardAttributes. See Table 1 for a description of each attribute.

Table 1. Attributes in TSAttribute Data Structure

Attribute Type	Attribute	Description
EBoardAttributes	BOARD_Parm_Act_Talk_En	enables or disables active talker on the physical board
EBoardAttributes	BOARD_Parm_ActTalkerNotify Interval	changes the default firmware interval for active talker notification events on the physical board. The value must be passed in 10 msec units. The default setting is 20 (0.2 seconds).
EBoardAttributes	BOARD_Parm_ToneClamping	enables or disables tone clamping to reduce the amount of DTMF tones heard on a per party basis on the physical board. The default setting is enabled.
EConferenceAttributes	CONF_Parm_DTMF_Mask	specifies a mask for the DTMF digits used for volume control. CNF_DIG values can be ORed to form the mask. For more information about CNF_DIG values, see CNF_VOL, on page 122.
EConferenceAttributes	CONF_Parm_ToneClamping	enables or disables DTMF tone clamping for the conference. Values are 0 to disable and 1 to enable. The default setting is enabled.
EPartyAttributes	PARTY_Parm_AGC_En	automatic gain control. Values are 0 to disable and 1 to enable. The default setting is disabled.
EPartyAttributes	PARTY_Parm_Broadcast_En	one party can speak while all other parties are muted. Values are 0 to disable and 1 to enable. The default setting is disabled.
EPartyAttributes	PARTY_Parm_Coach_En	party is set to coach. Coach is heard by pupil only. Values are 0 to disable and 1 to enable. The default setting is disabled.
EPartyAttributes	PARTY_Parm_Echo_Cancel	enables or disables echo cancellation for the party. Values are 0 to disable and 1 to enable. The default setting is disabled. The echo cancellation feature provides 128 taps (16 msec) of echo cancellation. **Note:** This attribute is only supported by **cnf_AddParty( )** and **cnf_AddPartyByTimeslot( )**. The **cnf_SetPartyAttributes( )** function cannot be used to set or modify echo cancellation.
EPartyAttributes	PARTY_Parm_Party_Mode	uses EPARTY_Parm_Party_Mode which is described below

Table 1. Attributes in TSAttribute Data Structure (Continued)

Attribute Type	Attribute	Description
EPartyAttributes	PARTY_Parm_Pupil_En	party is set to pupil. Pupil hears everyone including the coach. Values are 0 to disable and 1 to enable. The default setting is disabled.
EPartyAttributes	PARTY_Parm_Tariff_En	party receives periodic tone for duration of the call. Values are 0 to disable and 1 to enable. The default setting is disabled.
EPartyAttributes	PARTY_Parm_ToneClampingPty	DTMF tone clamping for the party. Values are 0 to disable and 1 to enable. The default setting is disabled.
EParty_Parm_Party_Mode	PARM_VAL_Party_Mode_FULL	indicates that the party may transmit and receive in the conference
EParty_Parm_Party_Mode	PARM_VAL_Party_Mode_NULL	indicates that no time slots are assigned. If this value is specified, no other party attributes can be specified. This is the default setting. This mode is used to reserve party resources for future use.
EParty_Parm_Party_Mode	PARM_VAL_Party_Mode_RCV	indicates that the party is in receive-only mode in the conference
EParty_Parm_Party_Mode	PARM_VAL_Party_Mode_XMIT	indicates that the party is in transmit-only mode in the conference

Example

For an example of the TSAttribute data structure, see the Example section of the **cnf_AddParty( )** and **cnf_SetPartyAttributes( )** function description.

TSAttributesList

```
typedef struct SAttributesList
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nAttrCount;
     TSAttribute *m_pAttrs;
} TSAttributesList;
```

Description

The TSAttributesList data structure specifies attributes for a party, a conference, or a physical board. This structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSAttributesList_VERSION_1.

m_nAttrCount
: specifies the number of elements in the m_pAttrs attributes array

m_pAttrs
: specifies an array of m_nAttrCount attributes for the party, conference, or physical board. For more information about attributes, see the data structure description for TSAttribute, on page 124.

Example

For an example of the TSAttributesList data structure, see the Example section of the **cnf_SetBoardAttributes( )**, **cnf_SetConferenceAttributes( )**, and **cnf_SetPartyAttributes( )** function description.

TSConferenceInfo

```
typedef struct SConferenceInfo
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyGroup;
     unsigned int m_nAttrCount;
     TSConferenceAttribute *m_pAttrs;
} TSConferenceInfo;
```

Description

The TSConferenceInfo data structure contains information about a conference. This data structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSConferenceInfo_VERSION_1.

m_nPartyGroup
: specifies the party group. Set to 0 if it doesn't matter which party group the conference will belong to. For a specific party group, specify the party group number in this field. The party group number is obtained from the data returned by **cnf_GetResourceCount( )**. For more information on party groups and resource management, see the *Conferencing API Programming Guide*.

m_nAttrCount
: specifies the number of elements in the m_pAttrs attributes array

m_pAttrs
: specifies an array of m_nAttrCount attributes for the conference. For more information, see the data structure description for TSAttribute, on page 124. (The TSConferenceAttribute data structure is an alias for TSAttribute.)

Example

For an example of the TSConferenceInfo data structure, see the Example section of the **cnf_CreateConference( )** function description.

TSCreateConferenceReply

```
typedef struct SCreateConferenceReply
{
     unsigned int m_nVersion; /* Version of this structure */
     int m_nConfHandle;
     char m_szConfName[MAX_CNFNAME + 1];
} TSCreateConferenceReply;
```

Description

The TSCreateConferenceReply data structure specifies conference information returned by the **cnf_CreateConference( )** function. This data structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSCreateConferenceReply_VERSION_1.

m_nConfHandle
: specifies the conference handle

m_szConfName
: specifies the conference name. The maximum length of the name is determined by the MAX_CNFNAME define, plus 1 for the NULL termination character. See *cnflib.h* for the MAX_CNFNAME definition.

Example

For an example of the TSCreateConferenceReply data structure, see the Example section of the **cnf_AttachConference( )** function description.

TSEventsList

```
typedef struct SEventsList
{
     unsigned int m_nVersion;     /* Version of this structure */
     unsigned int m_nEventCount;
     EListOfEvents *m_pEvents;
} TSEventsList;

typedef enum
{
     CNFENB_Event_DTMFDetection   /* triggers CNFEV_DTMF */
     CNFENB_Event_StatusChanged
} EListOfEvents;
```

Description

The TSEventsList data structure contains information about events. This data structure is used by the **cnf_EnableEvents( )** and **cnf_DisableEvents( )** functions. This structure is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSEventsList_VERSION_1.

m_nEventCount
: specifies the number of elements in the m_pEvents events array

m_pEvents
: specifies an array of m_nEventCount events. The ElistOfEvents data type is an enumeration that defines the following values:
  - CNFENB_Event_DTMFDetection – indicates that DTMF detection is enabled. The CNFEV_DTMF event is returned when the operation is complete.
  - CNFENB_Event_StatusChanged – indicates that the status of the event has changed. The events returned are CNFEV_STATUSADDPARTY or CNFEV_STATUSPARTYREMOVED.

Example

For an example of the TSEventsList data structure, see the Example section of the **cnf_EnableEvents( )** function description.

TSThirdPartyInfo

```
typedef struct SThirdPartyInfo
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyID;
     long m_nTxTimeslot;
     long m_nRxTimeslot;
     unsigned int m_nAttrCount;
     TSPartyAttribute *m_pAttrs;
} TSThirdPartyInfo;
```

Description

The TSThirdPartyInfo data structure contains information about the third party. This data structure is included in the TSThirdPartyList structure. It is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSThirdPartyInfo_VERSION_1.

m_nPartyID
: contains the party identification number (a returned value)

m_nTxTimeslot
: specifies the transmit time slot number of the third party

m_nRxTimeslot
: specifies the receive time slot number of the third party

m_nAttrCount
: specifies the number of elements in the m_pAttrs attributes array

m_pAttrs
: specifies an array of m_nAttrCount attributes for the third party. For more information, see the data structure description for TSAttribute, on page 124. The TSPartyAttribute data structure is an alias of TSAttribute.

Example

For an example of the TSThirdPartyInfo data structure, see the Example section of the **cnf_AddPartyByTimeslot( )** function description.

TSThirdPartyList

```
typedef struct SThirdPartyList
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyCount;
     TSThirdPartyInfo *m_pPartyInfo;
} TSThirdPartyList;
```

Description

The TSThirdPartyList data structure contains a count of the external parties and points to the TSThirdPartyInfo structure. It is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSThirdPartyList_VERSION_1.

m_nPartyCount
: specifies the number of elements in the m_pPartyInfo parties array

m_pPartyInfo
: specifies an array of m_nPartyCount parties. For more information, see the data structure description for TSThirdPartyInfo, on page 131.

Example

For an example of the TSThirdPartyList data structure, see the Example section of the **cnf_AddPartyByTimeslot( )** function description.

TSPartyGroupRes

```
typedef struct
{
     unsigned int m_nVersion;
     unsigned short m_nGrpMaxParties;
     unsigned short m_nGrpFreeParties;
     unsigned short m_nGrpMaxConferences;
     unsigned short m_nGrpFreeConferences;
} TSPartyGroupRes;
```

Description

The TSPartyGroupRes data structure contains the number of party resources and conference resources available in a party group. A party group is a collection of resources on the board. This structure is defined in *cnflib.h*.

For more information about party groups and resource management, see the *Conferencing API Programming Guide*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSPartyGroupRes_VERSION_1.

m_nGrpMaxParties
: specifies the maximum number of party resources available in the party group

m_nGrpFreeParties
: specifies the number of free party resources available in the party group

m_nGrpMaxConferences
: specifies the maximum number of conferences available in the party group

m_nGrpFreeConferences
: specifies the number of free conferences available in the party group

Example

For an example of the TSPartyGroupRes data structure, see the Example section of the **cnf_GetResourceCount( )** function description.

TSPartyInfo

```
typedef struct SPartyInfo
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyID;
     int m_nHandle;
     unsigned int m_nAttrCount;
     TSPartyAttribute *m_pAttrs;
} TSPartyInfo;
```

Description

The TSPartyInfo data structure contains information about the party. This data structure is included in the TSPartyList structure. It is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSPartyInfo_VERSION_1.

m_nPartyID
: contains the party identification number (a returned value)

m_nHandle
: contains the party handle, obtained from calls such as **dx_open( )**, **dt_open( )**, **gc_GetResourceH( )**, and **ipm_open( )**

m_nAttrCount
: specifies the number of elements in the m_nAttrCount attributes array

m_pAttrs
: specifies an array of m_nAttrCount attributes for the party. For more information, see the data structure description for TSAttribute, on page 124. The TSPartyAttribute data structure is an alias of TSAttribute.

Example

For an example of the TSPartyInfo data structure, see the Example section of the **cnf_AddParty( )** function description.

TSPartyList

```
typedef struct SPartyList
{
     unsigned int m_nVersion; /* Version of this structure */
     unsigned int m_nPartyCount;
     TSPartyInfo *m_pPartyInfo;
} TSPartyList;
```

Description

The TSPartyList data structure contains a count of the parties and points to the TSPartyInfo structure. It is defined in *cnflib.h*.

Field Descriptions

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

m_nVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. The current version of this data structure is TSPartyList_VERSION_1.

m_nPartyCount
: specifies the number of elements in the m_pPartyInfo parties array

m_pPartyInfo
: specifies an array of m_nPartyCount parties. For more information, see the data structure description for TSPartyInfo, on page 134.

Example

For an example of the TSPartyList data structure, see the Example section of the **cnf_AddParty( )** function description.

Error Codes 5

This chapter lists the error codes used by the conferencing software.

If a library function returns CNF_FAILURE to indicate failure, use the standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return the error description. These functions are described in the *Standard Runtime Library API Library Reference*.

The error codes used by the conferencing software are defined in *cnflib.h*. They are described as follows:

ECNF_BUFFERTOOSMALL
: buffer too small. Some conferencing functions point to buffers which can contain attributes, counts and so on. Be sure to allocate an appropriate buffer size.

ECNF_COMMANDNOTSUPPORTED
: function not supported on the specified device handle

ECNF_COMMANDTIMEDOUT
: function timed out while waiting for a response from the board

ECNF_DM3SUBSYSTEMERROR
: a severe internal error

ECNF_INVALIDARGUMENT
: invalid argument passed to a function

ECNF_INVALIDATTRIBUTE
: invalid attribute passed to a function

ECNF_INVALIDBRIDGE
: invalid bridge passed to **cnf_DeleteBridge( )**

ECNF_INVALIDCONFNAME
: invalid conference name passed to **cnf_AttachConference( )**

ECNF_INVALIDDEVICEHANDLE
: invalid device handle for a board, conference, or party

ECNF_INVALIDEVENT
: invalid event passed to **cnf_EnableEvents( )** or **cnf_DisableEvents( )**

ECNF_INVALIDPARTY
: invalid party passed to **cnf_AddParty( )**

ECNF_INVALIDPHYSICALBOARD
: invalid physical board name passed to **cnf_Open( )**

ECNF_INVALIDTIMESLOT
: invalid time slot passed to **cnf_AddPartyByTimeslot( )**

ECNF_MAXEDOUTRESOURCES
: system resources maxed out. The number of conferences or parties that can be added on a physical board have been exhausted.

ECNF_NULLPOINTERARGUMENT
one or more arguments to the function had an unexpected NULL value

ECNF_OUTOFMEMORY
internal memory allocation error indicating that system is low on memory

ECNF_TIMESLOTDOLERFAILED
TDM bus time slot allocation failed

Glossary

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

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

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

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

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

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

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

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

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

channel: 1. When used in reference to an Intel® Dialogic® analog expansion board, an audio path, or the activity happening on that audio path (for example, when you say the channel goes off-hook). 2. When used in reference to an Intel Dialogic digital expansion board, a data path, or the activity happening on that data path. 3. When used in reference to a bus, an electrical circuit carrying control information and data.

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

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

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

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

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

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

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

conferencing: Ability to perform a conference.

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

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

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

CPE: customer premises equipment.

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

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

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

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

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

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

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

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

DM3: Refers to Intel Dialogic mediastream processing architecture, which is open, layered, and flexible, encompassing hardware as well as software components. A whole set of products from Intel are built on DM3 architecture. Contrast with Springware which is earlier-generation architecture.

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

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

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

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

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

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

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

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

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

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

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

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

route: Assign a resource to a time slot.

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

SRL: See **Standard Runtime Library**.

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

Standard attribute functions for all Intel telecom devices are contained in the SRL. See standard runtime library (SRL).

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

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

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

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

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

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

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

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

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

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

Index

T

U

V