Modular Station Interface API for Linux and Windows Operating Systems

Library Reference

November 2003

05-1906-002

Publication Date: November 2003

Document Number: 05-1906-002

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 7

About This Publication 9

Purpose 9
Intended Audience 9
How to Use This Publication 9
Related Information 10

1 Function Summary by Category 11

1.1 Attribute Functions 11
1.2 Conference Management Functions 11
1.3 Configuration Functions 12
1.4 Device Management Functions 12
1.5 Diagnostic Functions 13
1.6 Extended Connection Functions 13
1.7 TDM Routing Functions 13
1.8 Station Functions 14
1.9 Extended Attribute Functions 15
1.10 MSI Function Support by Platform 15

2 Function Information 17

2.1 Function Syntax Conventions 17
ATMS_STATINFO( ) – retrieve information about the MSI board 18
ATMS_TSSGBIT( ) – retrieve the current station hook status 20
ms_addtoconf( ) – add one party to an existing conference 23
ms_chgxtder( ) – change the attribute of the connection extender 26
ms_close( ) – close the MSI device 29
ms_delconf( ) – delete a conference 31
ms_delxtdcon( ) – delete an extended connection 33
ms_dsprescount( ) – retrieve the available DSP resource count 35
ms_estconf( ) – establish a conference 38
ms_estxtdcon( ) – establish an extended connection 42
ms_genring( ) – generate ringing to a station 45
ms_genringCallerID( ) – send distinctive ring and caller ID information 50
ms_genringex( ) – generate distinctive ringing to a station 54
ms_genziptone( ) – generate a zip tone 60
ms_getbrdparm( ) – retrieve board parameters 62
ms_getcde( ) – retrieve the attributes of a participant 64
ms_getcnflist( ) – retrieve a conference list 67
ms_getctinfo( ) – retrieve device information 69
ms_getevt( ) – block and return control to the application 71
ms_getevtmsk( ) – retrieve station event mask 74
ms_getxmitslot( ) – return TDM bus time slot 76
ms_listen( ) – connect the receive of a station 78

ms_monconf( ) – add a monitor to a conference 81
ms_open( ) – open an MSI device. 84
ms_remfromconf( ) – remove a party from a conference 86
ms_ResultMsg( ) – retrieve an ASCII string describing a result code 89
ms_ResultValue( ) – retrieve the cause of an event 92
ms_SendData( ) – send data to station during a call. 95
ms_setbrdparm( ) – change board parameters 99
ms_setcde( ) – change the attributes of a party 106
ms_setevtmsk( ) – change transition event masks 109
ms_SetMsgWaitInd( ) – toggle message waiting indicator lamp 113
ms_setstparm( ) – change the MSI station level parameters 115
ms_setvol( ) – change or reset the station volume 117
ms_stopfn( ) – stop a multitasking function 119
ms_tstcom( ) – test the communication ability of a board 121
ms_tstdat( ) – perform a data test on the MSI board. 123
ms_unlisten( ) – disconnect the transmit and receive of a station. 126
ms_unmonconf( ) – remove a monitor from a conference. 128

3 Events 131

4 Data Structures 133

CT_DEVINFO – channel/station information. 134
MS_CADENCE – cadence information for distinctive ringing 137
MS_CDT – conference properties 139
MS_DataInfo – call waiting caller ID information 141
MS_NCB – notification tone characteristics 143
SC_TSINFO – TDM bus time slot information. 144

5 Error Codes 145

Glossary 149

Index 153

Figures

1 Ring Cadence Examples 104

Tables

1 MSI Function Support by Platform 15
2 MSI Board/Device Parameters 100
3 MSI Ring Cadence Examples 103
4 Ring Cadence Group A 138
5 Valid Attribute Combinations 140

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1906-002	November 2003	Global changes: Corrected description of MSMM_RNGOFFHK, MSMM_RNGSTOP, and MSMM_TERM; these are event data for MSEV_RING, not for MSEV_NORING. This change was made on the following function reference pages: ms_genring( ), ms_genringCallerID( ), and ms_genringex( ). Also, no longer show E_MSBADRNGSTA as event data for MSEV_RING and MSEV_NORING. (PTR 25051) In code examples, changed "tsinfo.sc_tsarray" to "tsinfo.sc_tsarrayp". This change was made on the following function reference pages: ms_addtoconf( ), ms_chgxtder( ), ms_estconf( ), ms_estxtdcon( ), ms_getxmitslot( ), ms_listen( ), ms_monconf( ), and ms_unmonconf( ). (PTR 29446) Changed mode from asynchronous to synchronous for the following functions: ms_getxmitslot( ), ms_listen( ), and ms_unlisten( ). (PTR 29710) ms_genringCallerID( ) function reference: Noted that the T:Time and Date sub-field used by the OrigAddr parameter is required as per Bellcore FR-NWT-00064 spec. (PTR 29371) CT_DEVINFO data structure reference: Provided updated and more detailed field descriptions. MS_CDT data structure reference: Added new values for the chan_attr field.
05-1906-001	January 2003	Initial version of document. Much of the information contained in this document was previously published in the *MSI Software Reference for UNIX and Windows*, document number 05-1218-003.

About This Publication

The following topics provide information about this publication:

- Purpose
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This publication provides a reference to all functions, parameters, and data structures in the Modular Station Interface (MSI) API library. It is a companion document to the *Modular Station Interface API Programming Guide*.

Intended Audience

This guide is intended for software developers who will access the modular station interface software. This may include any of the following:

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

How to Use This Publication

Refer to this publication after you have installed the hardware and the system software which includes the modular station interface software. This publication assumes that you are familiar with the Linux or Windows operating system and the C programming language.

The information in this guide is organized as follows:

- Chapter 1, "Function Summary by Category" groups the modular station interface library APIs into categories.
- Chapter 2, "Function Information" provides details about each modular station interface library function, including parameters, cautions, and error codes.
- Chapter 3, "Events" describes the events returned by the modular station interface library.

- Chapter 4, "Data Structures" provides details about each data structure used by the modular station interface library, including fields and descriptions.
- Chapter 5, "Error Codes" lists the error codes included in the modular station interface library.
- The Glossary provides definitions of key terms used in this document.

Related Information

For more information, refer to the following:

- *Modular Station Interface API Programming Guide*
- *Voice API Library Reference*
- *Voice API Programming Guide*
- *Standard Runtime Library API Library Reference*
- *Standard Runtime Library API Programming Guide*
- *http://developer.intel.com/design/telecom/support* (for technical support)
- *http://www.intel.com/design/network/products/telecom* (for product information)

Function Summary by Category 1

This chapter describes the categories into which the modular station interface (MSI) library functions can be logically grouped. This chapter also includes a table listing function support on various platforms (DM3, Springware).

- Attribute Functions 11
- Conference Management Functions 11
- Configuration Functions 12
- Device Management Functions 12
- Diagnostic Functions 13
- Extended Connection Functions 13
- TDM Routing Functions 13
- Station Functions 14
- Extended Attribute Functions 15
- MSI Function Support by Platform 15

1.1 Attribute Functions

These functions are used to retrieve specific information about the MSI board.

ms_dsprescount( )
returns DSP resource count

ms_getctinfo( )
returns information about the station interface device

1.2 Conference Management Functions

These functions are used to manage all conference activities.

ms_addtoconf( )
adds a party to an existing conference

ms_delconf( )
deletes a conference

ms_estconf( )
establishes a conference

ms_getcde( )
retrieves the attributes of a conference

ms_getcnflist( )
gets participant list

ms_monconf( )
adds a monitor to a conference

ms_remfromconf( )
removes a party from a conference

ms_setcde( )
sets the attributes of a conference

ms_unmonconf( )
removes a monitor from a conference

1.3 Configuration Functions

These functions set the MSI device or station level parameters and event masks, and check the status of the MSI device parameter settings.

ms_getbrdparm( )
returns board parameters

ms_getevt( ) (Windows only)
retrieves an unsolicited event

ms_getevtmsk( )
returns the station event mask

ms_ResultMsg( )
retrieves an ASCII string describing a result code

ms_ResultValue( )
retrieves the cause of an event

ms_setbrdparm( )
changes board parameters

ms_setevtmsk( )
changes station event mask

ms_setstparm( )
changes station level parameters

1.4 Device Management Functions

These functions are used to open and close devices. Before using any other MSI library function, the device must be opened to obtain the handle.

ms_close( )
closes an open MSI device

ms_open( )
opens an MSI device and returns a unique handle

ms_stopfn( )
stops a multitasking function in progress

1.5 Diagnostic Functions

Diagnostic functions check the functionality of the MSI firmware and hardware.

ms_tstcom( )
runs MSI communications test to determine whether the PC can communicate with the MSI board

ms_tstdat( )
runs data test on the MSI board to determine whether data is passed successfully between the PC and the MSI board

1.6 Extended Connection Functions

These functions are used to manage all extended connection activities.

ms_chgxtder( )
changes the attributes of the connection extender

ms_delxtdcon( )
deletes the extended connection

ms_estxtdcon( )
establishes an extended connection

1.7 TDM Routing Functions

TDM routing functions are used in time division multiplexing (TDM) bus configurations, which include the CT Bus and SCbus. A TDM bus is resource sharing bus that allows information to be transmitted and received among resources over multiple time slots.

TDM routing functions enable the application to make or break a connection between voice, telephone network interface, and other resource channels connected via TDM bus time slots. Each device connected to the bus has a transmit component that can transmit on a time slot and a receive component that can listen to a time slot.

The transmit component of each channel of a device is assigned to a time slot at system initialization and download. To listen to other devices on the bus, the receive component of the device channel is connected to any one time slot. Any number of device channels can listen to a time slot.

Note: When you see references to the SCbus or SCbus routing, this information also applies to the CT Bus. That is, the physical interboard connection can be either SCbus or CT Bus. The SCbus protocol is used and the SCbus routing API applies to all the boards regardless of whether they use an SCbus or CT Bus physical interboard connection.

A set of TDM routing functions exist for each Intel® Dialogic® library, such as fax (fx_ functions) and voice (dx_ functions). See the appropriate API Library Reference for more information on these functions.

ms_getxmitslot( )
returns the number of the TDM bus time slot connected to the transmit component of the station

ms_listen( )
connects the listen (receive) component of a station interface to a TDM bus time slot

ms_unlisten( )
disconnects the listen (receive) component of a station interface from a TDM bus time slot

Note: TDM routing convenience functions, **nr_scroute( )** and **nr_scunroute( )**, are provided to make or break a half- or full-duplex connection between any two channels transmitting on the bus. These functions are not a part of any library but are provided in a separate C source file called *sctools.c* located in the */usr/dialogic/sctools* directory. The functions are defined in *sctools.h*. MSI functionality may be conditionally compiled in or out of these functions using the SC_MSI and SC_DTI defines in the makefile provided. Refer to the *Voice API Library Reference* for more details.

1.8 Station Functions

These functions set individual characteristics to a station.

ms_genring( )
generates a ring to a station

ms_genringCallerID( )
sends distinctive ring and caller ID information

ms_genringex( )
generates distinctive ringing to a station

ms_genziptone( )
generates zip tone to a station

ms_SendData( )
sends data to station during a call

ms_SetMsgWaitInd( )
toggles message waiting indicator lamp

ms_setvol( )
sets station volume

1.9 Extended Attribute Functions

Extended Attribute functions return information specific to the MSI device specified in the function call.

ATMS_STATINFO( )
returns station information, including station number and location

ATMS_TSSGBIT( )
retrieves the current channel signaling bit status (on-hook or off-hook)

1.10 MSI Function Support by Platform

Table 1, "MSI Function Support by Platform", on page 15 provides an alphabetical listing of modular station interface API functions. The table indicates which platforms are supported for each of the functions. There are three platforms that use the MSI library: DI, HDSI, and Springware.

DM3 boards is a collective name used in this document to refer to products that are based on the Intel® Dialogic® DM3 mediastream architecture. Examples of DM3 boards that use the MSI library are Intel® NetStructure™ High Density Station Interface (HDSI) boards and Intel® Dialogic® Integrated Series boards.

Springware boards refer to boards based on earlier-generation architecture, such as the Intel® Dialogic® MSI-Global Series boards.

Although a function may be supported on both DM3 and Springware boards, there may be some restrictions on its use. For example, some parameters or parameter values may not be supported. For details, see the function reference descriptions in Chapter 2, "Function Information".

Table 1. MSI Function Support by Platform

Function	DI	HDSI	Springware
ATMS_STATINFO( )	S	S	S
ATMS_TSSGBIT( )	S	S	S
ms_addtoconf( )	S	NS	S
ms_chgxtder( )	NS	NS	S
ms_close( )	S	S	S
ms_delconf( )	S	NS	S
ms_delxtdcon( )	NS	NS	S
ms_dsprescount( )	S	S	S
ms_estconf( )	S	NS	S

Legend:
NS = Not Supported
S = Supported
* = Variance between platforms, refer to the function description for more information.

Table 1. MSI Function Support by Platform (Continued)

Function	DI	HDSI	Springware
ms_estxtdcon( )	NS	NS	S
ms_genring( )	S	S	S
ms_genringCallerID( )	S	S	NS
ms_genringex( )	S*	S*	S
ms_genziptone( )	NS	NS	S
ms_getbrdparm( )	S*	S*	S
ms_getcde( )	S	S	S
ms_getcnflist( )	S	NS	S
ms_getctinfo( )	S	S	S
ms_getevt( )	S	S	S
ms_getevtmsk( )	S	S	S
ms_getxmitslot( )	S	S	S
ms_listen( )	S	S	S
ms_monconf( )	S	NS	S
ms_open( )	S	S	S
ms_remfromconf( )	S	NS	S
ms_ResultMsg( )	S	S	NS
ms_ResultValue( )	S	S	NS
ms_SendData( )	S	S	NS
ms_setbrdparm( )	S*	S*	S
ms_setcde( )	S	NS	S
ms_setevtmsk( )	S	S	S
ms_SetMsgWaitInd( )	S	S	NS
ms_setstparm( )	S	S	S
ms_setvol( )	S	S	S
ms_stopfn( )	S	S	S
ms_tstcom( )	S	S	S
ms_tstdat( )	S	S	S
ms_unlisten( )	S	S	S
ms_unmonconf( )	S	NS	S

Legend:
NS = Not Supported
S = Supported
* = Variance between platforms, refer to the function description for more information.

Function Information 2

This chapter provides an alphabetical reference to the functions in the modular station interface library.

2.1 Function Syntax Conventions

The modular station interface functions use the following syntax:

```
data_type ms_function(device_handle, parameter1, ... parameterN)
```

data_type
: refers to the data type, such as integer, long, or void

ms_function
: represents the function name. Typically, modular station interface functions begin with “ms”. Extended Attribute functions begin with “ATMS.”

device_handle
: represents the device handle, which is a numeric reference to a device, obtained when a device is opened. The device handle is used for all operations on that device.

parameter1
: represents the first parameter

parameterN
: represents the last parameter

Note: Some MSI library functions can operate in either synchronous or asynchronous mode, using a **mode** parameter. Synchronous functions do not return control to the calling process until the function call is completed. When a function operates in asynchronous mode, the calling process retains control and a completion event is passed to the application to notify that the function is complete.

ATMS_STATINFO( )

Name: long ATMS_STATINFO (devh, statinfop)

Inputs: int devh — • MSI board device handle

char * statinfop — • pointer to four bytes containing station information

Returns: station information on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Attribute

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ATMS_STATINFO( )** function returns information about the board running the MSI software. This information includes the number and location of the stations on the board. The application is responsible for allocating the space (4 bytes) for the station information buffer.

Parameter	Description
devh	the valid MSI board device handle returned by a call to **ms_open( )**
statinfop	pointer to four bytes. When the function returns, the first byte contains the total number of stations on the board. For DM3 boards (DI and HDSI), byte 2 is fixed at 1 and bytes 3 and 4 are fixed at -1 (0xFF). For MSI boards, bytes 2, 3, and 4 indicate the status of the baseboard and two daughterboards, respectively.

Cautions

This function fails if an invalid device handle is specified. If no stations are present on the module, 0xFF is returned.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int i;
int  devh;                   /* Board device handle */
unsigned char statinfo[4];

/* Open board 1, device */
if ((devh = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1, errno=%d", errno);
    exit(1);
}

/*
 * Continue processing
 */125

/* Get board Ids and number of stations */
if ((ATMS_STATINFO(devh,statinfo)== -1){
    printf("Error getting station info\n");
    /* Close device and exit */
}

printf("Number of stations = %d\n",statinfo[0]);

for (i=0;i<4;i++){
   switch (statinfo[i]){
   case 0x01:
        printf("Board #%d present\n",i);
        break;
   case 0xff:
        printf("Board #%d not present\n",i);
        break;
   default:
        printf("Invalid module number %d\n",i);
        break;
   }
}

/*
 * Continue Processing
 */

 /* Done processing - close device */
if (ms_close(devh) == -1) {
    printf("Cannot close device msiB1. errno = %d", errno);
    exit(1);
}
```

See Also

None.

ATMS_TSSGBIT( )

Name: long ATMS_TSSGBIT (devh)

Inputs: int devh
- MSI station device handle

Returns: state of channel on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Attribute

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ATMS_TSSGBIT( )** function retrieves the current station hook status.

Parameter	Description
devh	the MSI station device handle returned by a call to **ms_open( )**

The returned bitmask represents the following:

MS_ONHOOK
MSI station is on-hook

MS_OFFHOOK
MSI station is off-hook

These equates are defined in *msilib.h*.

Cautions

This function fails if an invalid device handle is specified.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

/* Basic error handler */
do_error( devh, funcname )
   int devh;
   char *funcname;
{
   int errorval = ATDV_LASTERR( devh );
   printf( "Error while calling function %s.\n", funcname );
   printf( "Error value = %d.", errorval );
   printf( "\n" );
}

main()
{
   int tsdev;            /* Station device descriptor variable */
   long tsbits;          /* Time slot signaling bits */

   /*
    * Open board 1 channel 1 device
    */
   if ( ( tsdev = ms_open( "msiB1C1", 0 ) ) == -1 ) {
      printf( "Cannot open station msiB1C1. errno = %d", errno );
      exit( 1 );
   }

   /*
    * Get station signaling bits
    */
   tsbits = ATMS_TSSGBIT( tsdev );
   if ( tsbits == -1 ) {
      do_error( tsdev, "ATMS_TSSGBIT()" );
      exit( 1 );
   }

   switch(tsbits) {
      case MS_ONHOOK:
        /* continue processing (on-hook) */
        break;
      case MS_OFFHOOK:
        /* continue processing (off-hook) */
        break;
      default:
        printf("undefined parameter value = %d\n", tsbits);
        break;
   }

   /*
    * Continue processing
    *        .
    *        .
    *        .
    */

   /* Done processing - close device. */
   if ( ms_close( tsdev ) == -1 ) {
      printf( "Cannot close station msiB1C1. errno = %d", errno );
   }
}
```

See Also

None.

ms_addtoconf( )

Name: int ms_addtoconf (devh, confID, cdt)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier
MS_CDT *cdt	• pointer to conference descriptor table

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_addtoconf( )** function adds one party to an existing conference. The conference identifier specifies the conference to which the party will be added. When this function completes successfully, a party is added to a conference which causes a conferencing resource to be used. Only one party at a time can be added using this function.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier number
cdt	pointer to the conference descriptor table. See the MS_CDT data structure page for details.

In SCbus mode, this function returns the listen TDM bus time slot number for the MSPN_TS party. The number is placed in the MS_CDT structure for the MSPN_TS party. For an MSPN_STATION party, such information is not returned because the conferenced signal is not placed on the TDM bus. In SCbus mode, the chan_attr field in the MS_CDT structure is redefined as follows:

```
#define   chan_lts     chan_attr
```

Note: In SCbus mode, the MS_CDT structure is reused to return the listen TDM bus time slot information. The application is responsible for maintaining the integrity of the data in the structure.

Cautions

This function fails when:

- The device handle specified is invalid
- Too many parties are specified for a single conference

- The party is part of another conference
- The conference ID is invalid
- The board is out of DSP conferencing resources

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define     NUM_PARTIES          2

int  dev1;                    /* Board dev descriptor variables */
int  chdev2;                  /* Channel dev descriptor */
int  tsdev1, tsdev2;          /* Time slot dev desc */
MS_CDT cdt[NUM_PARTIES];      /* Conf. desc. table */
int  confID;                  /* Conf. ID */
SC_TSINFO    tsinfo;          /* Time slot info */
int  ts1, ts2;                /* SCbus time slots */
int  station;                 /* Station number */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
     printf( "Cannot open MSI B1: errno=%d", errno);
     exit(1);
}

/* Open board 1, channel 2 device */
if ((chdev2 = ms_open("msiB1C2",0) == -1) {
     printf("Cannot open MSIB1C2. errno = %d", errno);
     exit(1);
}

/* Assume MSI/SC is connected to a DTI via SCbus. */
/* Need to do a dt_open() for DTI time slots */
/* followed by dt_getxmitslot() to get SCbus time slots */
/* These SCbus time slots are passed on to the CDT */

/* ts1 & ts2 are used as the time slots */

/* Set up CDT structure */
cdt[0].chan_num = station ;  /* station is a valid station number */
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

/* SCbus time slot to be conferenced */
cdt[1].chan_num = ts1 ;              /* ts1 should be a valid time slot */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_NULL;
```

```
/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/* Do a listen for the TS */

tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

if (dt_listen(tsdev1, &tsinfo) == -1){
     printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
     exit(1);
}

/* Continue processing */

/* Add another party to conference */
cdt[0].chan_num = ts2;       /* ts2 should be a valid time slot */
cdt[0].chan_sel = MSPN_TS;
cdt[0].chan_attr = MSPA_RO|MSPA_TARIFF;

if (ms_addtoconf(dev1, confID,&cdt[0]) == -1) {
     printf("Error Message = %s",ATDV_ERRMSGP(dev1));
     exit(1);
}

/* Do a listen for the TS */

tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[0].chan_lts;

if (dt_listen(tsdev2, &tsinfo) == -1){
     printf("Error Message = %s",ATDV_ERRMSGP(tsdev2));
     exit(1);
}

/* Continue processing */
```

See Also

- ms_delconf( )
- ms_estconf( )
- ms_monconf( )
- ms_remfromconf( )
- ms_addtoconf( )

ms_chgxtder( )

Name: int ms_chgxtder (devh, xid, cdt)

Inputs:

int devh	• MSI board device handle
int xid	• extended connection identifier
MS_CDT *cdt	• pointer to descriptor table

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Connection

Mode: synchronous

Platform: Springware

Description

The **ms_chgxtder( )** function changes the attribute of the connection extender. After an extended connection has been established, only the channel attributes of the connection extender may be changed.

The signal that the connection extender should listen to is always present on the TDM bus, irrespective of the connection extender setting of the chan_sel field in the MS_CDT data structure.

Note: There can be only one connection extender per extended connection.

Parameter	Description
devh	the MSI board device handle
xid	the extended connection identifier
cdt	pointer to the conference descriptor table. See the MS_CDT data structure page for details.

Cautions

This function fails when:

- The device handle specified is invalid
- The board is not an MSI board
- The connection ID is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;            /* Device handle for board */
int  chdev2;          /* Station dev descriptor */
int  tsdev1,tsdev2;   /* DTI time slot device handles */
MS_CDT  cdt[3];       /* Connection descriptors */
int  xid;             /* Connection ID */
long lts;             /* listen time slot */
SC_TSINFO    tsinfo;  /* Time slot information structure */
int  rc;              /* Return Code */
int  station, ts1, ts2;

/* Start System */

/* Assume that there is a DTI in the system.
 * Assume two DTI transmit time slots. ts1 and
 * ts2, are identified by device handles tsdev1
 * and tsdev2, respectively.
 */

/*
 * Continue processing
 */
/*
 * Establish connection between a station and time slot ts1
 */
if ((rc=nr_scroute(tsdev1,SC_DTI,chdev2,SC_MSI,SC_FULLDUP))== -1) {
   printf("Error making connection between DTI timeslot\n");
   printf("and MSI station. rc = 0x%x\n",rc);
   exit(1);
}

/*
 * Now extend the connection established earlier
 */
cdt[0].chan_num = station ;      /* Use MSI station as connection identifier*/
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_PUPIL;

cdt[1].chan_num = ts2;           /* DTI time slot ts2 for connection extender */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_RO;

/* Establish extended connection. Since the extender is in receive only mode,
 * the connection will be extended without interrupting the conversation between the
 * external party and the station
 */
```

```
if (ms_estxtdcon(dev1,cdt,&xid) == -1) {
   printf("Error Message = %s",ATDV_ERRMSGP(dev1));
   exit(1);
}

/* Make tsdev2 listen to time slot returned by the ms_estxtdcon function  */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;
if (dt_listen(tsdev2,&tsinfo) == -1) {
   printf("Error Message = %s",ATDV_ERRMSGP(tsdev2));
   exit(1);
}

/* Prepare cdt to change the attribute of the connection extender */
cdt[0].chan_num = ts2 ;        /* Required station number */
cdt[0].chan_sel = MSPN_TS;
cdt[0].chan_attr = MSPA_COACH;

/* Change extender to coach */
if (ms_chgxtder(dev1,xid,cdt)== -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
```

See Also

- **ms_delxtdcon( )**
- **ms_estxtdcon( )**

ms_close( )

Name: int ms_close (devh)

Inputs: int devh
- MSI device handle

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Device Management

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_close( )** function closes the MSI device previously opened by the calling process and **ms_open( )**. The devices are either MSI boards or stations. The **ms_close( )** function releases the handle and breaks the link between the calling process and the device.

Parameter	Description
devh	the valid MSI device handle returned by a call to **ms_open( )**

Cautions

- This function fails if the device handle is invalid.
- The **ms_close( )** function affects only the link between the calling process and the device. Other processes are unaffected by **ms_close( )**.
- If event notification is active for the device to be closed, call the SRL **sr_dishdlr( )** function prior to calling **ms_close( )**.
- A call to **ms_close( )** does not affect the configuration of the MSI.
- Devices should **never** be closed using **close( )**.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>        /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

main()
{
   int bddev;               /* Board device descriptor variable */

   /* Open board 1 device */
   if ( ( bddev = ms_open( "msiB1", 0 ) ) == -1 ) {
      printf( "Cannot open board msiB1. errno = %d\n", errno );
      exit( 1 );
   }

   /*
    *   Continue processing
    *          .
    *          .
    *          .
    */

   /* Done processing - close device */
   if ( ms_close( bddev ) == -1 ) {
      printf( "Cannot close board msiB1. errno = %d", errno );
   }
}
```

See Also

- ms_open( )

ms_delconf( )

Name: int ms_delconf (devh, confID)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_delconf**( ) function deletes a conference previously established. The conference ID is the value previously returned by **ms_estconf**( ).

Parameter	Description
devh	the MSI board device handle
confID	the MSI conference identifier

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

2. It is the responsibility of the application to perform an unlisten for each party of the conference.

Cautions

None.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define    NUM_PARTIES    3

int  dev1;                   /* Board dev descriptor variables */
MS_CDT cdt[NUM_PARTIES];     /* Conf. desc. table */
int  confID;                 /* Conf. ID */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}

/*
 * Continue processing
 */

/* Set up CDT structure */
/* station 2, 4 and 7 are used to establish a conference */
cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

cdt[1].chan_num = 4;
cdt[1].chan_sel = MSPN_STATION;
cdt[1].chan_attr = MSPA_PUPIL;

cdt[2].chan_num = 7;
cdt[2].chan_sel = MSPN_STATION;
cdt[2].chan_attr = MSPA_COACH;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) != 0) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 * Continue processing
 */

if (ms_delconf(dev1, confID) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
/* Continue processing */
```

See Also

- **ms_addtoconf( )**
- **ms_estconf( )**
- **ms_monconf( )**
- **ms_remfromconf( )**
- **ms_unmonconf( )**

ms_delxtdcon( )

Name: int ms_delxtdcon (devh, xid)

Inputs:
- int devh — • MSI board device handle
- int xid — • extended connection identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Connection

Mode: synchronous

Platform: Springware

Description

The **ms_delxtdcon( )** function deletes an extended connection. The connection extender is removed on successful completion of this function. Calling this function does not affect the integrity of the connection. The two parties will still remain in a connection.

Parameter	Description
devh	the MSI board device handle
xid	the extended connection identifier number

Notes: ***1.*** It is the responsibility of the application to do an **ms_unlisten( )** for the connection extender.

2. Calling this function frees three resources.

Cautions

This function fails when:

- The device handle specified is invalid
- The device is not an MSI board
- The connection ID is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
int  dev1;                    /* Device handle for board */
int  xid;                     /* Connection ID */
SC_TSINFO   tsinfo;           /* Time slot information structure */

/* Start System */
/*
 * Assume that there is an extended connection between a
 * station and a time slot. xid is obtained from the previous
 * extended connection.
*/

/*
 *   Continue processing
 */

/*
 * Do an unlisten for the connection extender if it is a external
 * party
 */

/*
 * Delete the extended connection
 */
if (ms_delxtdcon(dev1,xid) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 * Continue processing
 */
```

See Also

- ms_chgxtder( )
- ms_estxtdcon( )

ms_dsprescount( )

Name: int ms_dsprescount (devh, valuep)

Inputs:

int *devh	• MSI board device handle
int *valuep	• pointer to the memory location to receive the free DSP resource count

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Attribute

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_dsprescount( )** function returns the available DSP resource count.

Parameter	Description
devh	the MSI board device handle
valuep	pointer to the location containing the free DSP resource count

Each DSP has a number of resources managed by the application. Calling certain MSI library functions may cause the available resource count to change. However, the channel selector of the party does not affect the resource usage. When zip tone support is enabled, one resource is used.

Note: On Springware boards, a conference is limited to eight parties. A monitor is counted as one of the eight parties.

The following MSI functions cause a change to the available resource count:

ms_addtoconf( )
Uses one resource every time a party is added to a conference

ms_remfromconf( )
Frees one resource

ms_delconf( )
Frees all resources in use by the conference

ms_delxtdcon( )
Frees three resources

ms_estconf( )
Uses the total number of parties in the conference

ms_estxtdcon( )
Uses three resources

ms_monconf( )
Uses one resource

ms_setbrdparm( )
When **parm_id** = MSG_ZIPENA and **value** = MS_ZIPENABLE, one resource will be used

When **parm_id** = MSG_ZIPENA and **value** = MS_ZIPDISABLE, one resource will be freed

ms_unmonconf( )
Frees one resource

Cautions

This function fails when the device handle specified is invalid.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;                    /* Board dev descriptor variables */
int  valuep;                  /* Resource count */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}
/* Get DSP resource count */
if (ms_dsprescount(dev1, &valuep) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

printf("Free DSP resource count = %d\n", valuep);

/*
 * Continue processing
 *
 */

if (ms_close(dev1)== -1){
    printf( "Cannot Close MSIB1: errno=%d", errno);
    exit(1);
}
```

See Also

- ms_addtoconf( )
- ms_delconf( )
- ms_delxtdcon( )
- ms_estconf( )
- ms_estxtdcon( )
- ms_monconf( )
- ms_remfromconf( )
- ms_setbrdparm( )
- ms_unmonconf( )

ms_estconf( )

Name: int ms_estconf (devh, cdt, numpty, confattr, confID)

Inputs:

int devh	• MSI board device handle
MS_CDT *cdt	• pointer to conference descriptor table
int numpty	• number of parties in a conference
int confattr	• conference attributes
int *confID	• pointer to memory location to receive the conference identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_estconf( )** function establishes a conference of up to four parties. **ms_addtoconf( )** must be used to increase the size of the conference beyond four and up to eight parties.

Parameter	Description
devh	MSI board device handle
cdt	pointer to the conference descriptor table. See the MS_CDT data structure page for details.
numpty	number of parties in the conference. When this function completes successfully, the conference is established and **numpty** resources are used.
confattr	bitmask describing the properties of the conference. These properties affect all parties in the conference. • MSCA_ND – All parties in conference are notified by a tone if another party is being added or removed from a conference. • MSCA_NN – If MSCA_ND is set, *do not* notify participants if a party joins the conference in "receive-only" mode or as a monitor. • MSCA_NULL – No special attributes. ***Note:*** The default MSCA_NULL must be used if the conference attribute is not specified.
confID	pointer to the memory location containing the conference ID number

In SCbus mode, this function returns the listen TDM bus time slot number for the MSPN_TS party. The number is placed in the MS_CDT structure for the MSPN_TS party. For an MSPN_STATION

party, such information is not returned because the conferenced signal is not placed on the TDM bus. In SCbus mode, the chan_attr field in the MS_CDT structure is redefined as follows:

```
#define   chan_lts     chan_attr
```

Note: In SCbus mode, the MS_CDT structure is reused to return the listen TDM bus time slot information. The application is responsible for maintaining the integrity of the data in the structure.

Cautions

- Parties to be added to a conference must be off-hook when **ms_estconf( )** is called.
- Cascading conferences are not supported in any form. A cascading conference occurs when the maximum number of eight participants are already joined in a conference and more participants are added. The cascading conference contains a time slot participant that is a time slot from a second monitored conference, therefore, creating a conference within a conference. Cascading conferences may significantly deteriorate conference voice quality.
- This function fails when:
  - An invalid device handle is specified
  - More than four parties are specified using **ms_estconf( )**
  - DSP resources are not available
  - Any of the parties specified are already in another conference on this device
  - Any of the stations specified are already listening to a TDM bus time slot

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define        NUM_PARTIES     3

int  dev1;                    /* Board dev descriptor variables */
int  chdev1,chdev2;           /* Channel dev descriptor */
MS_CDT cdt[NUM_PARTIES];      /* Conf. desc. table */
int  confID;                  /* Conf. ID */
int  ts1, ts2;

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}
```

```
/* Assume MSI/SC is connected to a DTI via SCbus. */
/* Need to do a dt_open() for DTI time slots */
/* This returns tsdev1 and tsdev2 as 2 device handles
/* for 2 time slots. Follow this by dt_getxmitslot()
/* to get SCbus time slots */
/* These SCbus time slots are passed on to the CDT */

/*
 * Continue processing
 */

/* Set up CDT structure */
/* Include station 2 on MSI board in conference */
cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

/* The chan_num below is the SCbus time slot for tsdev1 on which */
/* DTI time slot is transmitting. It is received as a result of  */
/* dt_getxmitslot() function above */
cdt[1].chan_num = ts1;
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_PUPIL;

/* Set up another SCbus time slot for tsdev2 to be part of a 3 party conference. Another DTI
time slot transmits on this SCbus time slot, just like above */

cdt[2].chan_num = ts2;
cdt[2].chan_sel = MSPN_TS;
cdt[2].chan_attr = MSPA_COACH;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) != 0) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/* Note no listen required for cdt[0] because it is a station */
/* Do a listen for cdt[1] */
/* Set up SC_TSINFO structure for SCbus tslot */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;

/* Now, listen to  TS */
if (dt_listen(tsdev1,&tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
    exit(1);
}

/* Do a listen for cdt[2] */
/* Set up SC_TSINFO structure for SCbus tslot */
tsinfo.sc_tsarrayp = &cdt[2].chan_lts;

/* Now, listen to  TS */
if (dt_listen(tsdev2,&tsinfo) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev2));
    exit(1);
}

/*
 * Continue processing
 *
 */
if (ms_delconf(dev1, confID) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
```

```
/* Continue processing  */
```

See Also

- ms_addtoconf( )
- ms_delconf( )
- ms_remfromconf( )

ms_estxtdcon( )

Name: int ms_estxtdcon (devh, cdt, xid)

Inputs:

int devh	• MSI board device handle
MS_CDT *cdt	• pointer to descriptor table
int *xid	• pointer to memory location containing the extended connection identifier

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Connection

Mode: synchronous

Platform: Springware

Description

The **ms_estxtdcon( )** function establishes an extended connection. An extended connection is a connection in which there is a third party. Calling this function uses three resources.

Parameter	Description
devh	the MSI board device handle
cdt	pointer to the conference descriptor table
xid	pointer to the memory location containing the extended connection number

For the purpose of this function, a connection is a full-duplex, TDM bus routing between two parties. A connection may be set up using the convenience function **nr_scroute( )**. It is the responsibility of the application to set up the connection prior to extending it. No verification of the presence of a connection between parties is made prior to extending the connection.

One party of the connection to be extended must be a station on the board for which the **ms_estxtdcon( )** function is issued. The other party is another station or a TDM bus time slot. Extended connections have a *connection extender* and a *connection identifier*. The differences are as follows:

- A connection extender is always the third party in a connection and can be either a station or a TDM bus time slot.
- A connection identifier must be a station. The attributes of the connection identifier can only be set at the time the extended connection is established.

Cautions

- Stations to be added to an extended connection must be off-hook when **ms_estxtdcon( )** is called.
- This function fails when:
  - The device handle specified is invalid
  - A prior connection has not been established
  - DSP resources are not available

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>    /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;              /* Device handle for board */
int  chdev2;            /* Station dev descriptor */
int  tsdev1,tsdev2;     /* DTI time slot device handles */
MS_CDT  cdt[3];         /* Connection descriptors */
int  xid;               /* Connection ID */
long lts;               /* listen time slot */
SC_TSINFO   tsinfo;     /* Time slot information structure */
int  rc;                /* Return Code */
int  station, ts1, ts2;
/* Start System */

/* Assume that there is a DTI in the system.
 * Assume two DTI transmit time slots. ts1 and
 * ts2, are identified by device handles tsdev1
 * and tsdev2, respectively.
 */
/*
 * Continue processing
 */

/*
 * Establish connection between a station and time slot ts1
 */
if ((rc=nr_scroute(tsdev1,SC_DTI,chdev2,SC_MSI,SC_FULLDUP))== -1) {
    printf("Error making connection between DTI time slot\n");
    printf("and MSI station. rc = 0x%x\n",rc);
    exit(1);
}
```

```
/*
 * Now extend the connection established earlier
 */
cdt[0].chan_num = station ;      /* Use MSI station as connection identifier*/
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_PUPIL;

cdt[1].chan_num = ts2;           /* DTI time slot ts2 for connection extender */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_RO;

/* Establish extended connection. Since the extender is in receive only mode,
 * the connection will be extended without interrupting the conversation between the
 * external party and the station
 */

if (ms_estxtdcon(dev1,cdt,&xid) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/* Make tsdev2 listen to time slot returned by the ms_estxtdcon function  */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &cdt[1].chan_lts;
if (dt_listen(tsdev2,&tsinfo) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev2));
    exit(1);
}

/* Prepare cdt to change the attribute of the connection extender */
cdt[0].chan_num = ts2 ;        /* Required station number */
cdt[0].chan_sel = MSPN_TS;
cdt[0].chan_attr = MSPA_COACH;

/* Change extender to coach */
if (ms_chgxtder(dev1,xid,cdt)== -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
```

See Also

- **ms_chgxtder( )**
- **ms_delxtdcon( )**

ms_genring( )

Name: int ms_genring (devh, len, mode)

Inputs:

int devh	• device handle for station
unsigned short len	• length in cycles for ring
unsigned short mode	• asynchronous/synchronous

Returns: 0 on success for asynchronous
>0 on success for synchronous
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: asynchronous or synchronous

Platform: DI, HDSI, Springware

Description

The **ms_genring( )** function generates ringing to a station. The function will terminate when the phone goes off-hook or the specified number of rings has been generated. **ms_genring( )** is only supported on boards that have ringing capability.

Parameter	Description
devh	the station device handle
len	the number of cycles to ring a station; a maximum value of 255 is allowed
mode	the operation mode For synchronous mode, EV_SYNC must be specified as the third parameter. The function will return only on termination of ringing due to an error, off hook, or completion of ring cycles. For asynchronous mode, EV_ASYNC must be specified as the third parameter. The function will return on initiation of ringing or on error. To get the completion status, a termination event is generated.

This function will use the default ring, which is the last distinctive ring that was generated on the station by the **ms_genringex( )** function, or if none, the board-level ring cadence as set by the **MSG_UDRNGCAD** parameter in the **ms_setbrdparm( )** function.

A ring duty cycle includes an on time (ring generation) and off time (no ring). If **ms_genring( )** is received by the MSI board during off time, ring generation will be delayed until the on time portion of the duty cycle is reached. This delay can be up to approximately four seconds. Specifying a

length, or cycle, of at least two rings is recommended to make sure that at least one full ring cycle is generated. If you specify one ring, the phone may not ring.

Note: For MSI/SC boards: When you ring a station, a built-in (non-modifiable) 500 ms ring is "splashed" to the station immediately before its ring cadence begins. The splash may make the beginning of the ring cadence sound slightly different from the rest of the cadence. This ring splash serves as a fast way to produce a ring at the station and, therefore, to reduce the glare window. Otherwise, glare could occur when a ring starts in the off-time (non-ringing) portion of the ring cycle (where there is no notification that the phone is being rung) and a person picks up the (silent) phone expecting to get dial tone and instead is connected with a caller.

Termination Events

When this function is called in asynchronous mode, a return value of 0 indicates that the function was initiated, while a return value of -1 indicates error. The following events may be received:

MSEV_RING
: Indicates successful completion of ring operation. The event data for MSEV_RING is:
  - MSMM_RNGOFFHK – Solicited off hook detected
  - MSMM_RNGSTOP – Ringing stopped by **ms_stopfn( )**
  - MSMM_TERM – Ringing terminated

MSEV_NORING
: Indicates the ring operation was not successful.

When this function is called in synchronous mode, a return value of -1 indicates failure and a positive return value (>0) indicates the reason for termination. Reasons for termination are:

MSMM_RNGOFFHK
: Solicited off hook detected

MSMM_RNGSTOP
: Ringing stopped by **ms_stopfn( )**

MSMM_TERM
: Ringing terminated

Cautions

- A glare condition occurs when two parties seize the same line for different purposes. If glare occurs in your application, the function returns successfully. However, it is followed by the event MSEV_NORING. The data associated with the event is E_MSBADRNGSTA, indicating that the station was off-hook when the ring was attempted.
- This function fails when:
  - Executed on a station currently off hook. The error returned is E_MSBADRNGSTA.
  - The MSI board does not support ringing capabilities
  - The device handle is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example - Synchronous

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;                   /* Station device descriptor */
int  rc;                     /* Return code */

/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSIBC11, station 1,channel 1: errno=%d", errno);
    exit(1);
}

/*
 * Continue processing
 */
/* Generate ringing for 10 cycles in sync mode*/
if ((rc =ms_genring(dev1,10,EV_SYNC)) == -1) {
    /* process error */
}

/* If timeout, process the condition */
if (rc=MSMM_TERM) {
    printf("Station not responding");
}

/*
 * Continue Processing
 */

 /* Done processing - close device */
if (ms_close(dev1) == -1) {
    printf("Cannot close device msiB1C1. errno = %d", errno);
    exit(1);
}
```

Example - Asynchronous

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;                   /* Station dev descriptor */
int  srlmode;                /* SRL mode indicator */
```

```
/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSIB1C1, station 1,channel 1: errno=%d", errno);
    exit(1);
}

/* Set up handler function to handle play completion */
if (sr_enbhdlr(dev1,MSEV_RING,sig_hdlr) == -1) {
      /* process error */
}

/*
 * Continue processing
 */
/* Generate ringing */
if (ms_genring(dev1,10,EV_ASYNC) == -1) {
    printf("Error could not set up ringing. Errno = %d", errno);
    exit(1);
}

/* On receiving the completion event, MSEV_RING, control is
   transferred to the handler function previously established
   using sr_enbhdlr().
*/

/*
 *   Continue Processing
 */

 /* Done processing - close device */
if (ms_close(dev1) == -1) {
    printf("Cannot close device msiB1C1. errno = %d", errno);
    exit(1);
}

/*
 * Continue processing
 */

int sig_hdlr()
{
   int   dev = sr_getevtdev();
   unsigned short *sigtype = (unsigned short *)sr_getevtdatap();

   if (sigtype != NULL) {
      switch (*sigtype) {
      case MSMM_TERM:
          printf("Station does not answer");
          return 0;

      case MSMM_RNGOFFHK:
          printf("Station offhook detected\n");
          return 0;

      default:
          return 1;
      }
   }

   /*
    * Continue processing
    */

}
```

See Also

- ms_genringCallerID( )
- ms_genringex( )
- ms_setbrdparm( )
- ms_setevtmsk( )
- ms_stopfn( )

ms_genringCallerID( )

Name: int ms_genringCallerID(devh, len, mode, Cadid, OrigAddr, rfu)

Inputs:

int devh	• station device handle
unsigned short len	• length in cycles for ring
unsigned short mode	• asynchronous/synchronous
unsigned short Cadid	• cadence ID or default ring
char* OrigAddr	• call origination information
void* rfu	• reserved for future use

Returns: 0 on success for asynchronous operation
>0 on success for synchronous operation
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: asynchronous or synchronous

Platform: DI, HDSI

Description

The **ms_genringCallerID( )** function allows transmission of analog caller ID data (call originator information) to telephones equipped with FSK caller ID detectors.

Parameter	Description
devh	device handle
len	ring length (in cycles)
mode	set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution
Cadid	Either a CadenceID for distinctive rings or the current default ring (MS_RNG_DEFAULT) may be specified
OrigAddr	ASCII character string that holds the information about the origination party. The maximum length is 64 characters. With this feature, **OrigAddr** can be divided into multiple sub-fields identified by field identifiers to hold analog caller identification (FSK) transmission data.
rfu	reserved for future use

The sub-fields used by the **OrigAddr** parameter include:

Caller Name
identifies the name of the call originator if available. Maximum value is 36 characters.

Caller Name Absence Reason
identifies why call originator's name is not available. Possible reasons are Private (P) or Out of Area (O).

Caller Number
identifies the number of the call originator if available.

Caller Number Absence Reason
identifies why call originator's number is not available. Possible reasons are Private (P) or Out of Area (O).

Date Time
identifies the date and time at which the call is sent, in the format: month, day, hour, minutes. For example, a string of T:01221215 would be interpreted as Jan 22, 12:15 PM.

Note: The T:Date Time sub-field is required as per Bellcore FR-NWT-00064 spec.

User Data
identifies that the data in this field is user-defined analog caller identification (FSK) data and it should be transmitted without parsing. This gives flexibility to the applications to transmit data that is not defined in the above sub-fields.

Sub-group identifiers with format **X:** are used to specify sub-fields for caller ID transmission. Note that the user strings embed this character as part of sub-field identifiers. Thus this sub-group identifier is implicit by nature.

The following sub-group identifiers are supported:

A:
identifies beginning of Caller Number Absence Reason sub-field.

B:
identifies beginning of Caller Name Absence Reason sub-field.

I:
identifies beginning of Caller Number sub-field.

N:
identifies beginning of Caller Name sub-field.

T:
identifies beginning of Time and Date sub-field. This sub-field is required as per Bellcore FR-NWT-00064 spec.

R:
identifies beginning of user-defined FSK data. This string holds the checksum data at the end. This field provides application support for transmission of FSK data that is not covered by the above fields.

Notes: ***1.*** Use the character '/' as an escape character to indicate that ':' is part of the string. For example, Next string "N:J/:NamathI:993-3000" uses the escape character / to embed the name J:Namath.

2. The end of a sub-field is recognized by the character ':' or the end of string when a sub-field is located at the end of the string.

Termination Events

When this function is called in asynchronous mode, a return value of 0 indicates that the function was initiated, while a return value of -1 indicates error. The following events may be received:

MSEV_RING
Indicates successful completion of ring operation. The event data for MSEV_RING is:
- MSMM_RNGOFFHK – Solicited off hook detected
- MSMM_RNGSTOP – Ringing stopped by **ms_stopfn( )**
- MSMM_TERM – Ringing terminated

MSEV_NORING
Indicates the ring operation was not successful.

When this function is called in synchronous mode, a return value of -1 indicates failure and a positive return value (>0) indicates the reason for termination. Reasons for termination are:

MSMM_RNGOFFHK
Solicited off hook detected

MSMM_RNGSTOP
Ringing stopped by **ms_stopfn( )**

MSMM_TERM
Ringing terminated

Cautions

None.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

The following code snippets demonstrate different uses of the **OrigAddr** parameter.

Application sends Caller Name and Date to a Caller ID phone connected to a station:

```
OriginationAddress[128] = "T:01310930N:John Doe" which is:
Caller Name = John Doe
Date Time = Jan 31, 9 30 am
```

Application sends Caller Number absence reason (O: Out Of Area) and Date and Time to a Caller ID phone connected to a station:

```
OriginationAddress[128] = "T:01310930A:O" which is:
Caller Number Absence Reason : Out Of Area
Date Time = Jan 31, 9 30 am
```

Application sends proprietary data to a Caller ID phone connected to a station:

```
OriginationAddress[128] = "T:01310930R:xxxxxxxxxxx" which is:
xxxx  represents the proprietary data to be sent
Date Time = Jan 31, 9 30 am (as per Bellcore FR-NWT-00064 spec, T:Time and Date is a mandatory
field)
```

See Also

- **ms_genring( )**
- **ms_genringex( )**
- **ms_SetMsgWaitInd( )**
- **ms_stopfn( )**

ms_genringex( )

Name: int ms_genringex (devh, len, mode, cadid)

Inputs:

int devh	• device handle for station
unsigned short len	• length in cycles for ring
unsigned short mode	• asynchronous/synchronous
unsigned short cadid	• cadence ID for distinctive ring

Returns: 0 on success for asynchronous
>0 on success for synchronous
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: asynchronous or synchronous

Platform: DI, HDSI, Springware

Description

The **ms_genringex( )** function generates distinctive ringing to a station. The function will terminate when the phone goes off hook or the specified number of rings has been generated. **ms_genringex( )** is only supported on boards with ringing capability.

A distinctive ring becomes that station's default ring. If you generate a distinctive ring on a station by setting the **cadid** parameter to a cadence ID, the specified distinctive ring becomes the default ring cadence for that station. Future rings generated either by the **ms_genringex( )** function when **cadid** is set to MS_RNG_DEFAULT or by the **ms_genring( )** function will use the default ring cadence. The default ring cadence is either the last distinctive ring that was generated on the station, or if none, the board-level ring cadence as set by the **MSG_UDRNGCAD** parameter in the **ms_setbrdparm( )** function.

Parameter	Description
devh	the station device handle
len	the number of cycles to ring a station. A maximum value of 255 is allowed.

Parameter	Description
mode	the operation mode For synchronous mode, EV_SYNC must be specified as the third parameter. The function will return only on termination of ringing due to an error, off hook, or completion of ring cycles. For asynchronous mode, EV_ASYNC must be specified as the third parameter. The function will return on initiation of ringing or on error. To get the completion status, a termination event is generated.
cadid	the cadence ID for distinctive ringing. Range: 1 - 8. See **ms_setbrdparm( )** MSG_DISTINCTRNG for information on initializing distinctive ring and assigning cadence IDs. ***Note:*** The following distinctive rings are not supported on DM3 boards: MS_RNGA_SPLASH3 and MS_RNGA_SPLASH4. Set **cadid** to MS_RNG_DEFAULT to use the default ring for that station. Rings generated either by the **ms_genring( )** function or by **ms_genringex( )** function when **cadid** is set to MS_RNG_DEFAULT will use the default ring cadence of 2 seconds ON, 4 seconds OFF. For MSI boards, you can change the default ring cadence length using the download parameter file, *RING.PRM* (see the DCM on-line help for more information).

A ring duty cycle includes an on time (ring generation) and off time (no ring). If **ms_genringex( )** is received by the MSI board during off time, ring generation will be delayed until the on time portion of the duty cycle is reached. This delay can be up to approximately four seconds. Specifying a length, or cycle, of at least two rings is recommended to make sure that at least one full ring cycle is generated. If you specify one ring, the phone may not ring.

Note: For MSI/SC boards: When you ring a station, a built-in (non-modifiable) 500 ms ring is "splashed" to the station immediately before its ring cadence begins. The splash may make the beginning of the ring cadence sound slightly different from the rest of the cadence. This ring splash serves as a fast way to produce a ring at the station and, therefore, to reduce the glare window. Otherwise, glare could occur when a ring starts in the off time (non-ringing) portion of the ring cycle (where there is no notification that the phone is being rung) and a person picks up the (silent) phone expecting to get dial tone and instead is connected with a caller.

Termination Events

When this function is called in asynchronous mode, a return value of 0 indicates that the function was initiated, while a return value of -1 indicates error. The following events may be received:

MSEV_RING

Indicates successful completion of ring operation. The event data for MSEV_RING is:

- MSMM_RNGOFFHK – Solicited off hook detected
- MSMM_RNGSTOP – Ringing stopped by **ms_stopfn( )**
- MSMM_TERM – Ringing terminated

MSEV_NORING

Indicates the ring operation was not successful.

When this function is called in synchronous mode, a return value of -1 indicates failure and a positive return value (>0) indicates the reason for termination. Reasons for termination are:

MSMM_RNGOFFHK
Solicited off hook detected

MSMM_RNGSTOP
Ringing stopped by **ms_stopfn( )**

MSMM_TERM
Ringing terminated

Cautions

- A glare condition occurs when two parties seize the same line for different purposes. If glare occurs in your application, the function returns successfully. However, it is followed by the event MSEV_NORING. The data associated with the event is E_MSBADRNGSTA, indicating that the station was off-hook when the ring was attempted.
- **ms_genringex( )** will fail when specifying an invalid **cadid** or if distinctive ring has not been initialized with the **ms_setbrdparm( )** MSG_DISTINCTRNG parameter. The error returned is E_MSBADRNGCAD.
- This function fails when:
  - Executed on a station currently off hook. The error returned is E_MSBADRNGSTA.
  - The MSI board does not support ringing capabilities.
  - The device handle is invalid.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example - Synchronous

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int        dev1;            /* Station device descriptor */
int        rc;              /* Return code */
MS_CADENCE cadence;
BYTE       pattern;

/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSIB1C1, station 1,channel 1: errno=%d", errno);
    exit(1);
}
```

```
/*
 * Setup distinctive cadence
 */
cadence.cadid = 1; /* First distinctive cadence */
cadence.cadlength = MS_RNGA_CADLEN;
pattern = MS_RNGA_TWOSEC;
cadence.cadpattern = &pattern; /* Pattern (secs) : 2 on 4 off */

/* Set 1st ring cadence to MS_RNGA_TWOSEC */
if (ms_setbrdparm(devh, MSG_DISTINCTRNG, (void *)&cadence)) == -1){
   printf("Error setting board parameter : %s\n",
           ATDV_ERRMSGP(devh));
   exit(1);
}

/*
 * Continue processing
 */

/* Generate ringing using distinctive ring 1 */
if ((rc =ms_genringex(dev1,10,EV_SYNC,1)) == -1) {
    /* process error */
}

/* If timeout, process the condition */
if (rc=MSMM_TERM) {
    printf("Station not responding");
}

/*
 * Continue Processing
 */

 /* Done processing - close device */
if (ms_close(dev1) == -1) {
    printf("Cannot close device msiB1C1. errno = %d", errno);
    exit(1);
}
```

Example - Asynchronous

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;                   /* Station dev descriptor */
int  srlmode;                /* SRL mode indicator */
MS_CADENCE cadence;
BYTE       pattern;

/* Open board 1, station 1 device */
if ((dev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSIB1C1, station 1,channel 1: errno=%d", errno);
    exit(1);
}

/* Set up handler function to handle play completion */
if (sr_enbhdlr(dev1,MSEV_RING,sig_hdlr) == -1) {
      /* process error */
}
```

```
/* Setup distinctive cadence 1 */
cadence.cadid = 1; /* First distinctive cadence */
cadence.cadlength = MS_RNGA_CADLEN;
pattern = MS_RNGA_TWOSEC;
cadence.cadpattern = &pattern; /* Pattern (secs) : 2 on 4 off */

/* Set 1st ring cadence to MS_RNGA_TWOSEC */
if (ms_setbrdparm(devh, MSG_DISTINCTRNG, (void *)&cadence)) == -1){
   printf("Error setting board parameter : %s\n", ATDV_ERRMSGP(devh));
   exit(1);
}

/*
 *   Continue processing
 */

/* Generate asynchronous ringing using distinctive ring 1 */
if ((rc = ms_genringex(dev1,10,EV_ASYNC,1)) == -1) {
    /* process error */
}

/* Use sr_waitevt to wait for the completion of ms_genring().
   On receiving the completion event, MSEV_RING, control is
   transferred to the handler function previously established
   using sr_enbhdlr().
*/

/*
 *   Continue Processing
 */

/* Done processing - close device */
if (ms_close(dev1) == -1) {
    printf("Cannot close device msiB1C1. errno = %d", errno);
    exit(1);
}

/*
 * Continue processing
 */

int sig_hdlr()
{
   int   dev = sr_getevtdev();
   unsigned short *sigtype = (unsigned short *)sr_getevtdatap();

   if (sigtype != NULL) {
      switch (*sigtype) {
         case MSMM_TERM:
             printf("Station does not answer");
                return 0;

         case MSMM_RNGOFFHK:
             printf("Station offhook detected\n");
                return 0;
             default:
                return 1;
      }
   }

   /*
    * Continue processing
    */
}
```

See Also

- ms_genring( )
- ms_genringCallerID( )
- ms_setbrdparm( )
- ms_setevtmsk( )
- ms_stopfn( )

ms_genziptone( )

Name: int ms_genziptone (devh)

Inputs: int devh • MSI station device handle

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: synchronous

Platform: Springware

Description

The **ms_genziptone( )** function generates a zip tone to the station associated with the device handle. A zip tone indicates an incoming call to agents using headsets.

The tone generated is defined by the zip tone block specified in the **ms_setbrdparm( )** function description. Tone will only be generated to an MSI station that is not part of a conference or routed to a TDM bus time slot.

Parameter	Description
devh	the valid MSI station device handle returned by a call to **ms_open( )**

Cautions

This function fails when:

- The station device handle is invalid.
- Zip tone is disabled.
- On DM3 boards, this function fails with error: “Can't map or allocate memory in driver.”

Errors

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

For information about error codes, refer to Chapter 5, “Error Codes”. Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  chdev1;                  /*Station dev descriptor variable */

/* Open  station 1 device */

if ((chdev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSIB1C1: errno=%d", errno);
    exit(1);
}

/* Generate Ziptone */
if (ms_genziptone(chdev1) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(chdev1));
    exit(1);
}

/* Close station 1 */

if ( ms_close(chdev1)) == -1) {
    printf( "Cannot Close MSIB1C1: errno=%d", errno);
    exit(1);
}
```

See Also

- **ms_setbrdparm( )**

ms_getbrdparm( )

Name: int ms_getbrdparm (devh, param, valuep)

Inputs:

int devh	• MSI device handle
unsigned long param	• device parameter defined name
void *valuep	• pointer to variable where the parameter value will be placed

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getbrdparm( )** function returns board parameters. Each parameter has a symbolic name that is defined in *dtilib.h* and *msilib.h*.

Parameter	Description
devh	valid device handle returned by a call to **ms_open( )**
param	parameter to be examined; see the **ms_setbrdparm( )** function description for details. ***Note:*** On DM3 boards, only MSG_RNG is supported for this function.
valuep	pointer to the variable where the parameter value will be returned

Cautions

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

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

main()
{
   int    devh;               /* MSI/SC board device descriptor */
   int    value;              /* Parameter value */
   int    cadence[8];         /* Ring cadence length and pattern */
   int    cadence_len;        /* Cadence active period length (in bytes) */

   if ((devh = ms_open("msiB1", 0)) == -1) {
      printf("Error opening msiB1 : errno = %d\n", errno);
      exit(1);
   }

   /* Determine board type : Ringing or Non-ringing */

   if (ms_getbrdparm(devh, MSG_RING, (void *)&value)) == -1) {
      printf("Error retrieving board parameter : %s\n ", ATDV_ERRMSGP(devh));
      exit(1);
   }

   if (value == MS_RNGBRD){
      printf("You have a ringing MSI/SC board\n");
   }
   else
      printf("You have a non-ringing MSI/SC board\n");

   /* Retrieve the board's ring-cadence pattern */

   if (ms_getbrdparm(devh, MSG_RNGCAD, (void *)&cadence[0]))== -1) {
      printf("Error retrieving board parameter : %s\n ", ATDV_ERRMSGP(devh));
      exit(1);
   }
   printf("The ring cadence is %d x 250ms long\n", cadence[0]);
   cadence_len = (cadence[0]+7)/8;

   for (index = 1; index <= cadence_len; index++) {
      printf("Active period cadence pattern is 0x%x\n", cadence[index]);
   }

   if (ms_close(devh) == -1) {
      printf("Error Closing msiB1 : errno = %d\n", errno);
      exit(1);
   }
   return;
}
```

See Also

- ms_setbrdparm( )

ms_getcde( )

Name: int ms_getcde (devh, confID, cdt)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier
MS_CDT *cdt	• pointer to MS_CDT structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getcde( )** function retrieves the attributes of a participant in an existing conference. This function requires that the participant's chan_num and chan_sel are specified in the MS_CDT structure. On successful completion of this function, the conference party attribute will be returned in the chan_attr field of the MS_CDT structure.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier
cdt	pointer to structure; see MS_CDT for details.

The chan_attr field of the MS_CDT structure is a bitmask that describes the party's properties within the conference. It is possible that a combination of any of the attributes will be returned.

Note: Invoke **ms_getcde( )** multiple times if the attributes of more than one party are desired.

Cautions

This function fails when:

- The device handle specified is invalid.
- An invalid conference ID is specified.
- The queried party is not in the conference.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define      NUM_PARTIES    2
int  dev1=1;                 /* Board dev descriptor variables */
MS_CDT  cdt[NUM_PARTIES];    /* Conf. desc. table */
int  confID;                 /* Conf. ID */
int  attr;                   /* Channel attribute */
int  station, ts;

/* Start the system */

/* Set up CDT structure */
cdt[0].chan_num = station ;  /* station is a valid station number */
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

/* SCbus time slot to be conferenced */
cdt[1].chan_num = ts ;        /* ts should be a valid time slot */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_NULL;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 *
 * Continue processing
 *
 */

/* Now get the attribute of MSI Station */
cdt[0].chan_num = station;      /* Station in the conference */
cdt[0].chan_sel = MSPN_STATION;

if( ms_getcde(dev1, confID, &cdt[0])== -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

attr = cdt[0].chan_attr;

/*
 * Continue Processing
 *
 */
```

See Also

- ms_setcde( )

ms_getcnflist( )

Name: int ms_getcnflist (devh, confID, numpty, cdt)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier
int *numpty	• pointer to the number of parties in the conference
MS_CDT *cdt	• pointer to conference descriptor table

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_getcnflist( )** function retrieves a conference list. The function returns the total number of parties within a conference, and information specific to each party in that conference. The party-specific information retrieved includes a party's channel, TDM bus time slot number, selector, and attribute description.

Note: The list is not returned in any specified order.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier
numpty	pointer to the party count
cdt	pointer to conference descriptor table; see MS_CDT for details. ***Note:*** The application is responsible for allocating an MS_CDT table with sufficient elements.

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

Cautions

This function fails when an invalid conference ID is specified.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>     /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  dev1;               /* Board dev descriptor variables */
int  partycnt;           /* Number of parties*/
MS_CDT  cdtp[8];         /* Conf. desc. table */
int  confID;             /* Conf. ID */
int  i;

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSIB1: errno=%d", errno);
    exit(1);
}

/* Get conference list */
if (ms_getcnflist(dev1, confID, &partycnt, &cdtp[0]) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

printf("Number of parties = %d\n", partycnt);

for (i=0; i<partycnt; i++){
    printf("Chan_num = %x", cdtp[i].chan_num);
    printf("Chan_sel = %x", cdtp[i].chan_sel);
    printf("Chan_att = %x', cdtp[i].chan_attr);
    }
if (ms_close(dev1)== -1){
    printf( "Cannot Close MSIB1: errno=%d", errno);
    exit(1);
}
```

See Also

- **ms_estconf( )**

ms_getctinfo( )

Name: int ms_getctinfo (devh, ct_devinfop)

Inputs:

int devh	• MSI station device handle
CT_DEVINFO *ct_devinfop	• pointer to information structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Attribute

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getctinfo( )** function retrieves information related to a station device on the MSI board.

Parameter	Description
devh	the station device handle
ct_devinfop	pointer to channel/station information structure. Upon function return, the structure contains device information. See CT_DEVINFO for details.

Cautions

This function fails if an invalid station handle is specified.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  devh;                    /* Time slot device handle */
CT_DEVINFO  ct_devinfo;       /* Device information structure */
```

```
/* Open board 1 station 1 device */
if ((devh = ms_open("msiB1C1", 0)) == -1) {
    printf("Cannot open station msiB1C1.  errno = %d", errno);
    exit(1);
}

/* Get Device Information */
if (ms_getctinfo(devh, &ct_devinfo) == -1) {
    printf("Error message = %s", ATDV_ERRMSGP(devh));
    exit(1);
}

printf("%s Product Id = 0x%x, Family = %d, Network = %d, Bus mode = %d, Encoding = %d",
            ATDV_NAMEP(devh), ct_devinfo.ct_prodid, ct_devinfo.ct_devfamily,
            ct_devinfo.ct_nettype, ct_devinfo.ct_busmode, ct_devinfo.ct_busencoding);
```

See Also

- **ag_getctinfo( )**
- **dx_getctinfo( )**
- **dt_getctinfo( )**

ms_getevt( )

Name: int ms_getevt (devh, eblkp, timeout)

Inputs:

int devh	• MSI device handle
EV_EBLK * eblkp	• pointer to event block
int timeout	• time-out value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getevt( )** function is supported under Windows only. It blocks and returns control to the application for a specified event. This happens after one of the unsolicited events set by **ms_setevtmsk( )** occurs on the station device specified by the **devh** parameter or if a time-out occurs.

Parameter	Description
devh	the valid device handle returned by a call to **ms_open( )**
evtblkp	pointer to the event that ended the blocking
timeout	the maximum amount of time to wait for an event to occur. If **timeout** is set to -1, the **ms_getevt( )** function does not time out and blocks until an event occurs. If **timeout** is set to 0 and an event is not present, the function returns immediately with a -1 return code.

On successful return from the function, the event block structure, EV_EBLK, will have the following information:

ev_dev
The device on which the event occurred. This is the same as the **devh** passed to the function.

ev_event
MSEV_SIGEVT indicating signaling transition event.

ev_data
An array of bytes where ev_data[0] and ev_data[1] contain the signaling information. Signaling information is retrieved in short variable. Refer to the example below for information on retrieving this data.

The event block structure is defined as follows:

```
typedef struct ev_eblk {
      int ev_dev;                /* Device on which event occurred */
      unsigned long ev_event;    /* Event type */
      int ev_len;                /* Length of data associated with event */
      unsigned char ev_data[8];  /* 8 byte data buffer */
      void ev_datap;             /* variable pointer if more than 8 bytes of data */
} EV_EBLK;
```

Cautions

- The **ms_getevt( )** function is not supported under the Linux operating system.
- This function fails when:
  - The device handle is invalid for an MSI device
  - The event field is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

EV_EBLK eblk;

main()
{
   int devh;                  /* Board device handle */
   unsigned short sigmsk = MSMM_ONHOOK | MSMM_OFFHOOK | MSMM_HOOKFLASH;
   short sig;

  /*
   * Open station 1 device
   */

   if ((devh = ms_open("msiB1C1",0)) == -1) {
      printf("Error: Cannot open board 1 station 1. errno = 0x%x\n",errno);
      exit(1);
   }

   if (ms_setevtmsk(devh, MSEV_SIG, sigmsk, DTA_SETMSK) == -1) {
       printf("%s: ms_setevtmsk MSEV_SIGMSK DTA_SETMSK ERROR %d: %s:Mask = 0x%x\n",
                   ATDV_NAMEP(devh),ATDV_LASTERR(devh),ATDV_ERRMSGP(devh),sigmsk);
       ms_close(devh);
       exit(1);
   }
```

```
   /*
    *  Wait for events on this time slot
    */
   while(1) {
      ms_getevt ( devh, &eblk, -1 );    /* Wait forever */
      if (eblk.ev_event == MSEV_SIGEVT) {
         sig = eblk.ev_data[0]  |   (short) eblk.ev_data[1] << 8 ;
         if ((sig & MSMM_ONHOOK) == MSMM_ONHOOK)
             printf("Onhook signal received\n");
         if ((sig & MSMM_OFFHOOK) == MSMM_OFFHOOK)
             printf("Offhook signal received\n");
         if ((sig & MSMM_HOOKFLASH) == MSMM_HOOKFLASH)
             printf("Hook flash signal received\n");
      }
   } /* end of while statement */
}
```

See Also

- **ms_getevtmsk( )**

ms_getevtmsk( )

Name: int ms_getevtmsk (devh, event, bitmaskp)

Inputs:

int devh	• MSI station device handle
int event	• event to retrieve
unsigned short *bitmaskp	• pointer to bitmask variable

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getevtmsk( )** function returns a station event mask for a specified event.

Parameter	Description
devh	the valid station device handle returned by a call to **ms_open( )**
event	specifies an event's mask: • MSEV_SIGMSK – hook switch transition event (on-hook transition event, off-hook transition event, hookflash event)
bitmaskp	pointer to a variable that will contain the value of the bitmask. Refer to **ms_setevtmsk( )** for the valid bitmask values.

Cautions

This function fails when:

- The device handle is invalid for an MSI station device
- The event field is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

/* Basic error handler */
do_error( devh, funcname )
   int devh;
   char *funcname;
{
   int errorval = ATDV_LASTERR( devh );
   printf( "Error while calling function %s.\n", funcname );
   printf( "Error value = %d.", errorval );
   printf( "\n" );
}

main()
{
   int tsdev;                 /* Station device descriptor variable */
   unsigned short bitmask;    /* Bitmask variable */

   /* Open board 1 device */
   if ( ( tsdev = ms_open( "msiB1C1", 0 ) ) == -1 ) {
      printf( "Cannot open board msiB1C1. errno = %d", errno );
      exit( 1 );
   }
   /* Get signaling event mask*/
   if ( ms_getevtmsk( tsdev, MSEV_SIGMSK, &bitmask ) == -1 ) {
      do_error( tsdev, "ms_getevtmsk( )");
   }

   if ( bitmask & MS_ONHOOK) {
      /* continue processing (ON-HOOK event is set) */
      printf("ON-HOOK event is set\n");
   }

   if ( bitmask & MS_OFFHOOK) {
      /* continue processing (OFF-HOOK event is set) */
      printf("OFF-HOOK event is set\n");
   }

   if ( bitmask & MS_HOOKFLASH) {
      /* continue processing (HOOK FLASH event is set) */
      printf("HOOK FLASH event is set\n");
   }

   /*
    *          .
    *   Continue processing
    *          .
    */

   /* Done processing - close device */
   if ( ms_close( tsdev ) == -1 ) {
      printf( "Cannot close board msiB1C1. errno = %d", errno );
   }
}
```

See Also

- ms_setevtmsk( )

ms_getxmitslot( )

Name: int ms_getxmitslot (devh, tsinfop)

Inputs:

int devh	• MSI station device handle
SC_TSINFO *tsinfop	• pointer to TDM bus time slot information structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: TDM Bus Routing

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_getxmitslot( )** function returns the TDM bus time slot of the station transmit channel.

Parameter	Description
devh	the station device handle
tsinfop	pointer to the TDM bus time slot information structure, see SC_TSINFO for details. Upon successful return from the function, SC_TSINFO contains the number of TDM bus time slots connected to the transmit of the station and points to the array that contains the TDM bus time slots (between 0 and 1023).

Note: The transmit of an MSI station device can be connected to only one external TDM bus time slot.

Cautions

This function fails when an invalid station device handle is specified.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "errno.h"

int       chdev;                 /* Station dev descriptor */
SC_TSINFO tsinfo;                /* Time slot information structure */
long      scts;                  /* SCbus time slot */

/* Open board 1, station 1 device */
if ((chdev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSI B1, C1: errno=%d", errno);
    exit(1);
}

/* Set up SC_TSINFO structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;

/* Get time slot on which MSI board 1, channel 1 is transmitting */
if (ms_getxmitslot(chdev1,&tsinfo) == -1) {
    printf("Error message = %s", ATDV_ERRMSGP(chdev));
    exit(1);
}

printf("msiB1C1 is transmitting on SCbus time slot %d",scts);
```

See Also

- ms_listen( )

ms_listen( )

Name: int ms_listen (devh, tsinfop)

Inputs:

int devh	• MSI station device handle
SC_TSINFO *tsinfop	• pointer to TDM bus time slot information structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: TDM Bus Routing

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_listen( )** function connects the receive channel of a station to a TDM bus time slot.

Parameter	Description
devh	the board device handle
tsinfop	pointer to the TDM bus time slot information structure, see SC_TSINFO for details.

Notes: ***1.*** An MSI station device can listen to one and only one time slot at a time; however, multiple devices may listen to the same time slot at the same time.

2. It is recommended that you issue a call to **ms_unlisten( )** before invoking this function.

Cautions

- The **ms_listen( )** function can take up to 30 msec to execute when running under the Linux operating system. On the Windows operating system, this function executes in approximately 1 msec.
- This function fails when:
  - An invalid station handle is specified
  - The TDM bus time slot number is invalid
- By default, MSI stations do not drive audio data onto their assigned TDM bus time slot. MSI stations only transmit audio data after receiving an **ms_listen( )** command. An application requiring an MSI station to transmit audio data without listening to another device (as in half-duplex mode) can have the MSI station listen to its own TDM bus time slot. This is accomplished as shown in the code sample below:

```
#include <windows.h>      /* For Windows applications only */
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "errno.h"

int  chdev1;              /* Channel dev descriptor */
int  tsdev1;              /* Time slot dev desc */
SC_TSINFO  tsinfo;        /* Time slot info     */
long scts;                /* SCbus time slot    */

/* Open board 1, channel 1 device */
if ((chdev1 = ms_open("msiB1C1",0) == -1) {
    printf("Cannot open MSIB1C1. errno = %d", errno);
    exit(1);
}

if ((tsdev1 = dt_open("dtiB1T1",0) == -1) {
    printf("Cannot open dtiB1T1. errno = %d", errno);
    exit(1);
}

/* Setup SC_TSINFO structure  */

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

/* Get transmit time slot of MSI/SC device chdev1*/
if (ms_getxmitslot(chdev1, &tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(chdev1));
    exit(1);
}

/* Make chdev1 listen to itself */
if (ms_listen(chdev1, &tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(chdev1));
    exit(1);
}

/* chdev1 is now transmitting on its SCbus time slot. Now, make tsdev1 listen to chdev1 */
if (dt_listen(tsdev1, &tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
    exit(1);
}

/* Continue processing */
if (dt_close(tsdev1) == -1){
    printf("Could not Close msiB1C1. errno = %d", errno);
    exit(1);
}
if (ms_close(chdev1) == -1){
    printf("Could not Close msiB1C1. errno = %d", errno);
    exit(1);
}
```

Errors

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

For information about error codes, refer to Chapter 5, “Error Codes”. Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>           /* For Windows applications only */
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"
#include "errno.h"

int       chdev1, tsdev4;      /* Chan dev descriptor variables */
SC_TSINFO tsinfo;              /* Time slot information structure */
long       scts;               /* SCbus time slot */

/* Open board 1, channel 1 device */
if ((chdev1 = ms_open("msiB1C1",0)) == -1) {
    printf( "Cannot open MSI B1, C1: errno=%d", errno);
    exit(1);
}
/* Open board 1, time slot 4 device */
if ((tsdev1 = dt_open("dtiB1T4",0)) == -1) {
    printf( "Cannot open DTI B1, T4: errno=%d", errno);
    exit(1);
}
/* Set up SC_TSINFO structure */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &scts;

/* Get time slot on which DTI board 1, time slot 4 is transmitting */
if (dt_getxmitslot(tsdev4,&tsinfo) == -1) {
    printf("Error message = %s", ATDV_ERRMSGP(chdev));
    exit(1);
}

/* Make MSI board 1, station 1 listen to transmit time slot
     of DTI Board 1 time slot 4 on SCbus */
if (ms_listen(chdev1,&tsinfo) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev4));
    exit(1);
}
```

See Also

- ms_getxmitslot( )
- ms_unlisten( )

ms_monconf( )

Name: int ms_monconf (devh, confID, lts)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier
long *lts	• pointer to listen TDM bus time slot

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_monconf**( ) function adds a monitor to a conference. Monitoring a conference guarantees that the conferenced signal will be placed on the TDM bus. This is slightly different from when a receive-only party is added to a conference. In case of a receive-only party, the conferenced signal may or may not be placed on the TDM bus, depending on the chan_sel of the party. Since the monitored signal is on the TDM bus, several parties can listen to the monitored signal simultaneously. It is the application's responsibility to listen to the TDM bus time slot on which the monitored signal is transmitted.

A monitor counts as one of the parties in the conference. If the maximum number of parties allowed is used, it is not possible to monitor the conference. When a conference is deleted, the conference monitor is also deleted.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier
lts	pointer to the listen TDM bus time slot; the monitored signal will be present on this time slot

Notes: ***1.*** This function can only be issued once per conference. If you attempt to add another monitor using **ms_monconf**( ), you will receive the E_MSMONEXT error message.

2. Calling this function uses one resource.

Cautions

This function fails when:

- The device handle specified is invalid
- The conference is full
- The board is out of DSP resources
- The conference ID is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define       NUM_PARTIES      2

int  dev1;                  /* Board dev descriptor variables */
int  tsdev1;                /* DTI time slot device handle */
MS_CDT  cdt[NUM_PARTIES];   /* Conf. desc. table */
int  confID;                /* Conf. ID */
long  lts;                  /* listen time slot */
SC_TSINFO   tsinfo;         /* Time slot information structure */
int  ts1;

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}

/* Assume that there is a DTI in the system.
 * Assume the device handle for a time slot on the DTI
 * is tsdev1 and time slot it is assigned to is ts1
*/

/* Set up CDT structure */
cdt[0].chan_num = station ;       /* Valid MSI Station */
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr =MSPA_NULL;

cdt[1].chan_num = ts1;           /* ts1 is a valid DTI time slot */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr =MSPA_TARIFF;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 * Continue Processing
 */
```

```
/* Now monitor the conference */

if (ms_monconf(dev1, confID,&lts) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
/* Assume that a DTI device tsdev1 is available  */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &lts;
if (dt_listen(tsdev1,&tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
    exit(1);
}

/*
 * Continue Processing
 */
```

See Also

- ms_unmonconf( )

ms_open( )

Name: int ms_open (name, oflags)

Inputs: char *name — • MSI station or board device name

int oflags — • open attribute flags

Returns: device handle
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Device Management

Mode: synchronous

Platform: DI, HDSI, Springware

■ Description

The **ms_open( )** function opens an MSI device and returns a unique handle to identify the device. All subsequent references to the opened device must be made using the device handle.

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

Parameter	Description
name	points to an ASCIIZ string that contains the name of a valid MSI station or board device The name of the station device should be **msiBbCc** where: • **b** is the board number (1 based) • **c** is the station number (1 to 24) The name of the board device should be **msiBb** where: • **b** is the board number (1 based)
oflags	reserved for future use. Set this parameter to 0.

■ Cautions

- Dialogic devices should never be opened using **open( )**.
- This function fails when:
  - The device name is invalid
  - The device is already open
  - The system has insufficient memory to complete the open

Errors

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

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

main()
{
   int bddev;               /* Board device descriptor variable */

   /* Open board 1 device */
   if ( ( bddev = ms_open( "msiB1", 0 ) ) == -1 ) {
      printf( "Cannot open board msiB1. errno = %d\n", errno );
      exit( 1 );
   }

   /*
    *   Continue processing
    *          .
    *          .
    *          .
    */

   /* Done processing - close device */
   if ( ms_close( bddev ) == -1 ) {
      printf( "Cannot close board msiB1. errno = %d", errno );
   }
}
```

See Also

- ms_close( )

ms_remfromconf( )

Name: int ms_remfromconf (devh, confID, cdt)

Inputs:

int devh	• MSI board device handle
int confID	• conference identifier
MS_CDT *cdt	• pointer to MS_CDT structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_remfromconf( )** function removes a party from a conference. The conference ID is the value previously returned by the **ms_estconf( )** function. In this case, the channel attributes of the MS_CDT structure are ignored.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier number
cdt	pointer to the conference descriptor table. See the MS_CDT data structure page for details.

Notes: *1.* It is recommended to call **ms_unlisten( )** before removing the TDM bus time slot member.

2. Calling this function frees one resource.

Cautions

- An error will be returned if this function is used to remove the last remaining party from a conference. The **ms_delconf( )** function must be used to end a conference.
- This function fails when:
  - The device handle passed is invalid
  - The conference ID is invalid
  - The party to be removed is not part of the specified conference

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>        /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define        NUM_PARTIES    3

int  dev1;                  /* Board dev descriptor variables */
MS_CDT cdt[NUM_PARTIES];    /* Conf. desc. table */
int  confID;                /* Conf. ID */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}

/*
 *  Continue processing
 */

/* Set up CDT structure */
/* Assume  MSI stations 2, 4 and 7 are in the conference */
cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

cdt[1].chan_num = 4
cdt[1].chan_sel = MSPN_STATION;
cdt[1].chan_attr = MSPA_PUPIL;

cdt[2].chan_num = 7;
cdt[2].chan_sel = MSPN_STATION;
cdt[2].chan_attr = MSPA_COACH;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) != 0) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 * Continue processing
 *
 */

cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
```

```
if (ms_remfromconf(dev1, confID, &cdt[0]) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

if (ms_delconf(dev1, confID) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);

}

/* Continue processing */
```

See Also

- **ms_addtoconf( )**
- **ms_delconf( )**
- **ms_estconf( )**

ms_ResultMsg( )

Name: int ms_ResultMsg(devh, result_code, msg)

Inputs:

int devh	• station device handle
long result_code	• error code
char **msg	• pointer to address of error code description

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: Synchronous

Platform: DI, HDSI

Description

The **ms_ResultMsg( )** function returns an ASCII string which describes a result code.

Parameter	Description
devh	device handle for station
result_code	error code. This parameter can be an error code returned by **ms_ResultValue( )**
msg	pointer to address where the description of the **result_code** message is stored.

Cautions

- Do not overwrite the ***msg** pointer since it points to private internal MSI data space.
- This function fails when an invalid **devh** or **msg** parameter is passed.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>    /* For Windows application only */
#include <stdio.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;          /* Station Device Descriptor    */
int rc;            /* Return Code                  */

long EventHandler (unsigned long temp)
{

    int dev=sr_getevtdev();
    long event=sr_getevttype();
    void* datap = (void*) sr_getevtdatap();
    char *errorMsg;
    long errorCode = 0;

    switch(event)
    {
    case MSEV_DATASENT :
        printf("Received MSEV_DATASENT for device = %s... \n",ATDV_NAMEP(dev));
        /* Continue processing */
        break;

    case MSEV_SENDDATAFAILED :
        ms_ResultValue(dev,event,datap,&errorCode);
        ms_ResultMsg(dev,errorCode,&errorMsg);
        printf("Received MSEV_SENDDATAFAILED for device = %s...ErrorCode =
                0X%X ErrorMessage = %s\n", ATDV_NAMEP(dev),errorCode,errorMsg);
        /* Continue processing */
        break;

    default :
        printf("Unknown event received on %s...Event = 0x%x Device = %d\n",
                ATDV_NAMEP(dev),event,dev);
        /* Continue processing */
        break;

    }   /* switch event ends */

    /* Continue processing */
    return 0;
}   /* EventHandler ends */

MS_DataInfo myDataInfo;
/*    DataString    Caller Name = John Doe
            Date Time = Jan 31, 9 30 am */
char DataString[128] = "T:01310930N:John Doe";

/* Open board 1, Station 1 device */
if ( (dev1 = ms_open("msiB1C1", 0)) == -1)
{
    printf("Cannot open msiB1C1, Station 1, Channel 1: errno=%d\n",errno);
    exit(1);
}

/* Set up handler function */
if (sr_enbhdlr(dev1, EV_ANYEVT, &EventHandler) == -1)
{
    /* process error */
}
```

```
/*
*    Continue processing
*    make sure the station is already in a call
*/

/*    Send data to a station which is already in a call in ASYNC mode    */
myDataInfo.version=0;
myDataInfo.dataType=eMSFSK;
myDataInfo.uInfo.dataString=DataString;

if((rc=ms_SendData(dev1,myDataInfo,EV_ASYNC))==-1)
{
    /* process error    */
}

/* Use sr_waitevt to wait for the completion of ms_SendData().
   On receiving the completion event, MSEV_DATASENT /  MSEV_SENDDATAFAILED
   control is transferred to the handler function (EventHandler)
   previously established using sr_enbhdlr().
*/

/*
*    Continue processing
*/

/* Done processing - close device */
if(ms_close(dev1)==-1)
{
    printf("Cannot close device msiB1C1. errno=%d\n",errno);
    exit(1);
}
```

See Also

- **ms_ResultValue( )**

ms_ResultValue( )

Name: int ms_ResultValue(devh, event, eventInfo, resultValue)

Inputs:

int devh	• station device handle
long event	• event identifier
void *eventInfo	• pointer to eventInfo block
long *resultValue	• address of error code value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: Synchronous

Platform: DI, HDSI

Description

The **ms_ResultValue( )** function is used to retrieve the cause of an event.

Parameter	Description
devh	device handle for station
event	event identifier
eventInfo	pointer to eventInfo block. Retrieve the pointer by calling the Standard Runtime Library function **sr_getevtdatap( )**. See the *Standard Runtime Library API Library Reference* for details.
resultValue	address where the MSI result value is stored

Cautions

- This function will fail if an invalid **devh**, **event**, or **eventInfo** parameter is passed.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>    /* For Windows application only */
#include <stdio.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;          /* Station Device Descriptor    */
int rc;            /* Return Code                  */

long EventHandler (unsigned long temp)
{

    int dev=sr_getevtdev();
    long event=sr_getevttype();
    void* datap = (void*) sr_getevtdatap();
    char *errorMsg;
    long errorCode = 0;

    switch(event)
    {
    case MSEV_DATASENT :
        printf("Received MSEV_DATASENT for device = %s... \n",ATDV_NAMEP(dev));
        /* Continue processing */
        break;

    case MSEV_SENDDATAFAILED :
        ms_ResultValue(dev,event,datap,&errorCode);
        ms_ResultMsg(dev,errorCode,&errorMsg);
        printf("Received MSEV_SENDDATAFAILED for device = %s...ErrorCode =
                0X%X ErrorMessage = %s\n", ATDV_NAMEP(dev),errorCode,errorMsg);
        /* Continue processing */
        break;

    default :
        printf("Unknown event received on %s...Event = 0x%x Device = %d\n",
                ATDV_NAMEP(dev),event,dev);
        /* Continue processing */
        break;

    }   /* switch event ends */

    /* Continue processing */
    return 0;
}   /* EventHandler ends */

MS_DataInfo myDataInfo;
/*    DataString    Caller Name = John Doe
            Date Time = Jan 31, 9 30 am */
char DataString[128] = "T:01310930N:John Doe";

/* Open board 1, Station 1 device */
if ( (dev1 = ms_open("msiB1C1", 0)) == -1)
{
    printf("Cannot open msiB1C1, Station 1, Channel 1: errno=%d\n",errno);
    exit(1);
}

/* Set up handler function */
if (sr_enbhdlr(dev1, EV_ANYEVT, &EventHandler) == -1)
{
    /* process error */
}
```

```
/*
*    Continue processing
*    make sure the station is already in a call
*/

/*    Send data to a station which is already in a call in ASYNC mode    */
myDataInfo.version=0;
myDataInfo.dataType=eMSFSK;
myDataInfo.uInfo.dataString=DataString;

if((rc=ms_SendData(dev1,myDataInfo,EV_ASYNC))==-1)
{
    /* process error    */
}

/* Use sr_waitevt to wait for the completion of ms_SendData().
   On receiving the completion event, MSEV_DATASENT /  MSEV_SENDDATAFAILED
   control is transferred to the handler function (EventHandler)
   previously established using sr_enbhdlr().
*/

/*
*    Continue processing
*/

/* Done processing - close device */
if(ms_close(dev1)==-1)
{
    printf("Cannot close device msiB1C1. errno=%d\n",errno);
    exit(1);
}
```

See Also

- **ms_ResultMsg( )**

ms_SendData( )

Name: int ms_SendData(devh, dataInfo, mode)

Inputs:

int devh	• station device handle
MS_DataInfo dataInfo	• pointer to information structure
int mode	• asynchronous/synchronous

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: asynchronous or synchronous

Platform: DI, HDSI

Description

The **ms_SendData( )** function sends data to a station while the station is already in conversation. For example, this function can be used to send call waiting caller ID information for a new call while a call is already in progress.

Parameter	Description
devHandle	device handle of the station
dataInfo	pointer to information structure; see MS_DataInfo for details.
mode	set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution

Termination Events

The following events may be returned when this function is called in asynchronous mode:

MSEV_DATASENT
Indicates the data was sent successfully to the specified station.

MSEV_SENDDATAFAILED
Indicates the send data operation failed.

Cautions

- This function will fail if executed when the station is on-hook. Use **ms_genringCallerID( )** to send caller ID information to the station when the station is on-hook.
- To avoid a situation where the other end of a call can hear the FSK tones being sent, your application should call **ms_unlisten( )** before calling **ms_SendData( ).**

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Possible errors for this function include:

EDT_BADDEV
: Bad device handle

EDT_BADGLOB
: Bad global parameter number

EDT_PARAMERR
: Invalid command parameter

EDT_INVTS
: Invalid device specified

E_MSINVDATATYPE
: Invalid data type specified while sending data to the station

E_MSINVVERSION
: Invalid version number specified

E_MSNOCNT
: Station not connected

Example - Synchronous

```
#include <windows.h>    /* For Windows application only */
#include <stdio.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;          /* Station Device Descriptor    */
int rc;            /* Return Code                  */

MS_DataInfo myDataInfo;
/*    DataString    Caller Name = John Doe
            Date Time = Jan 31, 9 30 am */
char DataString[128] = "T:01310930N:John Doe";

/* Open board 1, Station 1 device */
if ( (dev1 = ms_open("msiB1C1", 0)) == -1)
{
    printf("Cannot open msiB1C1, Station 1, Channel 1: errno=%d\n",errno);
    exit(1);
}

/*
*    Continue processing
*    make sure the station is already in a call
*/
```

```
/*    Send data to a station which is already in a call in SYNC mode    */
myDataInfo.version=0;
myDataInfo.dataType=eMSFSK;
myDataInfo.uInfo.dataString=DataString;
if((rc=ms_SendData(dev1,myDataInfo,EV_SYNC))==-1)
{
    /* process error    */
}

/*
*    Continue processing
*/

/* Done processing - close device */
if(ms_close(dev1)==-1)
{
    printf("Cannot close device msiB1C1. errno=%d\n",errno);
    exit(1);
}
```

Example - Asynchronous

```
#include <windows.h>    /* For Windows application only */
#include <stdio.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;          /* Station Device Descriptor    */
int rc;            /* Return Code                  */

long EventHandler (unsigned long temp)
{

    int dev=sr_getevtdev();
    long event=sr_getevttype();
    void* datap = (void*) sr_getevtdatap();
    char *errorMsg;
    long errorCode = 0;

    switch(event)
    {
    case MSEV_DATASENT :
        printf("Received MSEV_DATASENT for device = %s... \n",ATDV_NAMEP(dev));
        /* Continue processing */
        break;

    case MSEV_SENDDATAFAILED :
        ms_ResultValue(dev,event,datap,&errorCode);
        ms_ResultMsg(dev,errorCode,&errorMsg);
        printf("Received MSEV_SENDDATAFAILED for device = %s...ErrorCode =
                0X%X ErrorMessage = %s\n", ATDV_NAMEP(dev),errorCode,errorMsg);
        /* Continue processing */
        break;

    default :
        printf("Unknown event received on %s...Event = 0x%x Device = %d\n",
                ATDV_NAMEP(dev),event,dev);
        /* Continue processing */
        break;

    }   /* switch event ends */
```

```
    /* Continue processing */
    return 0;
}   /* EventHandler ends */

MS_DataInfo myDataInfo;
/*    DataString    Caller Name = John Doe
            Date Time = Jan 31, 9 30 am */
char DataString[128] = "T:01310930N:John Doe";

/* Open board 1, Station 1 device */
if ( (dev1 = ms_open("msiB1C1", 0)) == -1)
{
    printf("Cannot open msiB1C1, Station 1, Channel 1: errno=%d\n",errno);
    exit(1);
}

/* Set up handler function */
if (sr_enbhdlr(dev1, EV_ANYEVT, &EventHandler) == -1)
{
    /* process error */
}

/*
*    Continue processing
*    make sure the station is already in a call
*/

/*    Send data to a station which is already in a call in ASYNC mode    */
myDataInfo.version=0;
myDataInfo.dataType=eMSFSK;
myDataInfo.uInfo.dataString=DataString;

if((rc=ms_SendData(dev1,myDataInfo,EV_ASYNC))==-1)
{
    /* process error    */
}

/* Use sr_waitevt to wait for the completion of ms_SendData().
   On receiving the completion event, MSEV_DATASENT /  MSEV_SENDDATAFAILED
   control is transferred to the handler function (EventHandler)
   previously established using sr_enbhdlr().
*/

/*
*    Continue processing
*/

/* Done processing - close device */
if(ms_close(dev1)==-1)
{
    printf("Cannot close device msiB1C1. errno=%d\n",errno);
    exit(1);
}
```

See Also

- **ms_genringCallerID( )**
- **ms_unlisten( )**

ms_setbrdparm( )

Name: int ms_setbrdparm (devh, param, valuep)

Inputs:

int devh	• MSI board device handle
unsigned long param	• device parameter defined name
void * valuep	• pointer to parameter value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_setbrdparm( )** function is used to change board parameters.

Parameter	Description
devh	valid board device handle returned by a call to **ms_open( )**
param	parameter whose value is to be altered. Table 2 contains a description of MSI device parameters, listed alphabetically. *Note:* On DM3 boards, only MSG_DISTINCTRNG is supported for the **ms_setbrdparm( )** function.
valuep	void pointer to location containing the parameter value

Note: Calling this function may cause the available resource count to change as follows:

When parm_id = MSG_ZIPENA and value = MS_ZIPENABLE, one resource will be used.

When parm_id = MSG_ZIPENA and value = MS_ZIPDISABLE, one resource will be freed.

Cautions

- Most parameter values are integers. However, because this routine expects a void pointer to **valuep**, the address must be cast as a void*.
- This function fails when:
  - The device handle is invalid
  - The parameter specified is invalid

Table 2. MSI Board/Device Parameters

Parameter ID	Description
MSG_DBOFFTM	Defines the minimum length of time (50 ms units) before an off-hook transition is detected. Off-hook debounce time range: 0-3CH, where 0 = the value used to disable the off-hook debounce; default = 3H. A pointer to a short containing this duration is passed as the **valuep** parameter. **Note:** This parameter is not supported on DM3 boards.
MSG_DBONTM	Defines the minimum length of time (50 ms units) before an on-hook transition is detected. On-hook debounce time range: 5-3CH, default: 1EH. A pointer to a short containing this duration is passed as the third parameter. The MSG_DBONTM value must be set to a greater unit than MSG_MAXFLASH. If set to a lesser unit, the unit will automatically be made equal to or 1 unit greater than MSG_MAXFLASH. **Note:** This parameter is not supported on DM3 boards.
MSG_DISTINCTRNG	For **ms_setbrdparm( )**, this parameter initializes distinctive ringing and associates a cadence ID with a user-defined distinctive ring cadence. The cadence ID can then be used in the **ms_genringex( )** function. **Note:** This parameter is supported on DM3 boards for the **ms_setbrdparm( )** function only. For **ms_getbrdparm( )**, this parameter returns a filled-in MS_CADENCE structure containing the distinctive ring cadence pattern and length for the cadence ID specified in the **cadid** field. When initializing distinctive ringing using the **ms_setbrdparm( )** function, the **valuep** parameter must point to a data structure. For details, see the description of MS_CADENCE, on page 137. **Note:** Distinctive ring and board-level ring cadence are mutually exclusive except in the case where the cadence lengths are identical. You will get an E_MSRNGCADCNFLCT error if the MSG_PDRNGCAD or MSG_UDRNGCAD board-level ring cadence is currently set to a cadence that does not match the distinctive ring cadence length. For example, if MSG_UDRNGCAD is set to a cadence length of 4, you cannot initialize distinctive ring Group A, which uses a length of 6 seconds.
MSG_MAXFLASH	Defines a maximum length of time (50 ms units) for a station to be in an on-hook state before a hook flash signal is detected. Maximum hook flash time range: 5-3CH, default = 14H. A pointer to a short containing this duration is passed as the third parameter. **Note:** This parameter is not supported on DM3 boards.
MSG_MINFLASH	Defines a minimum length of time (50 ms units) for a station to be in an on-hook state before a hook flash signal is detected. Minimum hook flash time range: 2-14H, default = 6H. A pointer to a short containing this duration is passed as the third parameter. **Note:** This parameter is not supported on DM3 boards.

Table 2. MSI Board/Device Parameters (Continued)

<table>
<tr><th>Parameter ID</th><th colspan="3">Description</th></tr>
<tr><td>MSG_PDRNGCAD</td><td colspan="3">This parameter is used to select one of the following predefined ring cadence patterns on the MSI board. The default value (in seconds) is 6.
Note: This parameter is not supported on DM3 boards.</td></tr>
<tr><td></td><td>Value</td><td colspan="2">Cadence Pattern</td></tr>
<tr><td></td><td>1</td><td>1 on</td><td>5 off</td></tr>
<tr><td></td><td>2</td><td>1 on</td><td>2.75 off</td></tr>
<tr><td></td><td>3</td><td>1.5 on</td><td>3 off</td></tr>
<tr><td></td><td>4</td><td>1 on</td><td>4.25 off</td></tr>
<tr><td></td><td>5</td><td>.5 on, 2.5 off</td><td>.5 on, 2.5 off</td></tr>
<tr><td></td><td>6</td><td>2 on</td><td>4 off</td></tr>
<tr><td></td><td colspan="3">Note: Board-level ring cadence and distinctive ring are mutually exclusive except in the case where the cadence lengths are identical. You will get an E_MSRNGCADCNFLCT error if a distinctive ring is currently initialized through MSG_DISTINCTRNG and you set a MSG_PDRNGCAD board-level ring cadence with a length that does not match the distinctive ring length. For example, when using distinctive ring Group A, which has a cadence length of 6-seconds, you cannot set MSG_PDRNGCAD to the predefined cadence patterns that do not have a 6-second cycle.</td></tr>
<tr><td>MSG_RING</td><td colspan="3">This parameter is used to find out whether the board supports ringing capabilities. For a ringing board, the parameter value returned is MS_RNGBRD and for a non-ringing board, the parameter value is MS_NORNGBRD.
Note: This parameter is supported on DM3 boards for the ms_getbrdparm( ) function only.</td></tr>
<tr><td>MSG_RNGCAD</td><td colspan="3">This parameter is used to get the ring cadence pattern. The length of this parameter, in bytes, is variable and is determined by the number of bits of the active period cadence information specified. The first byte of this parameter, specifies the total number (count) of cadence bits being specified. A zero value for this first byte indicates the default number of bits (currently 8) is being specified. The next byte(s) correspond to the bit pattern(s).
Note: This parameter is not supported on DM3 boards.</td></tr>
</table>

Table 2. MSI Board/Device Parameters (Continued)

Parameter ID	Description
MSG_UDRNGCAD	User defined ring cadence: This parameter is used to set the default board-level ring cadence (the repeating pattern of ringing ON/OFF durations to a user-defined value for all stations attached to an MSI board). **Note:** This parameter is not supported on DM3 boards. The ring cadence is 1/3 active and 2/3 inactive. The active pattern defines an ON/OFF sequence of ringing in units of 250 ms and is specified in the value pointed to by **valuep**. The value can be from 2 to 7 bytes, depending upon the duration of the active cycle and subject to the active cycle length, which can be modified through the downloadable parameter file, *RING.PRM* (MSI boards only). Byte 1 specifies the total number of bits in the active period, ranging from 04H to 18H (4 - 24 bits). Since each bit represents a 250 ms duration, the active period can range from 1 second to 6 seconds. Bytes 2 - 7 (the number of bytes depends upon the value specified in byte 1) specifies the active period ringing pattern, with each bit representing the state of the ring current (1=ON, 0=OFF) for a 250 ms duration in a sequence from left to right (high-order bits before low-order bits). The inactive cycle is a mandatory period of no ringing that is twice the active cycle duration. It is not specified by the user but is created implicitly from the active cycle duration. The default active ring cycle is specified by the *RING.PRM* download parameter file (MSI boards only). If no download parameter file is used, the default active cycle is 2 seconds. Also see Table 3, “MSI Ring Cadence Examples”, on page 103 and Figure 1, “Ring Cadence Examples”, on page 104. **Note:** When you want to implement a given cadence pattern, you can use the following formula to determine how much to pad the active period with a trailing off-time period so that the total off time is correct. (For multiple ring cadence patterns, calculate the on-time as the entire period from the beginning of the first on-time to the end of the last on-time.) trailing-off-time = 1/3 total-off-time – 2/3 on-time or trailing-off-time = (total-off-time – (2 * on-time))/3 **For the MSI/SC-R boards only:** Even though the ring cadence may be defined as beginning with a ring ON, the **ms_genringex( )** or **ms_genring( )** function may ring a station beginning at any point in the ring cycle. Board-level ring cadence and distinctive ring are mutually exclusive except in the case where the cadence lengths are identical. You will get an E_MSRNGCADCNFLCT error if a distinctive ring is currently initialized through MSG_DISTINCTRNG and you set a MSG_UDRNGCAD board-level ring cadence with a length that does not match the distinctive ring length. For example, when using distinctive ring Group A, which has a cadence length of 6 seconds, you cannot set MSG_UDRNGCAD to a cadence length other than 6 seconds. If a user defines a new cadence with a greater active cycle length during run time, the current system wide total cycle length is increased to match it. When this occurs, all other defined cadences are padded with silence. System wide total cycle lengths cannot be reduced during run time. The active cycle length may be reduced during system initialization via the *RING.PRM* file which, in turn, will reduce the total cycle length (MSI boards only).
MSG_ZIPENA	The zip tone setting. MS_ZIPENABLE enables zip tone generation. MS_ZIPDISABLE disables zip tone generation. **Default** = MS_ZIPENABLE. **Note:** This parameter is not supported on DM3 boards.

Table 2. MSI Board/Device Parameters (Continued)

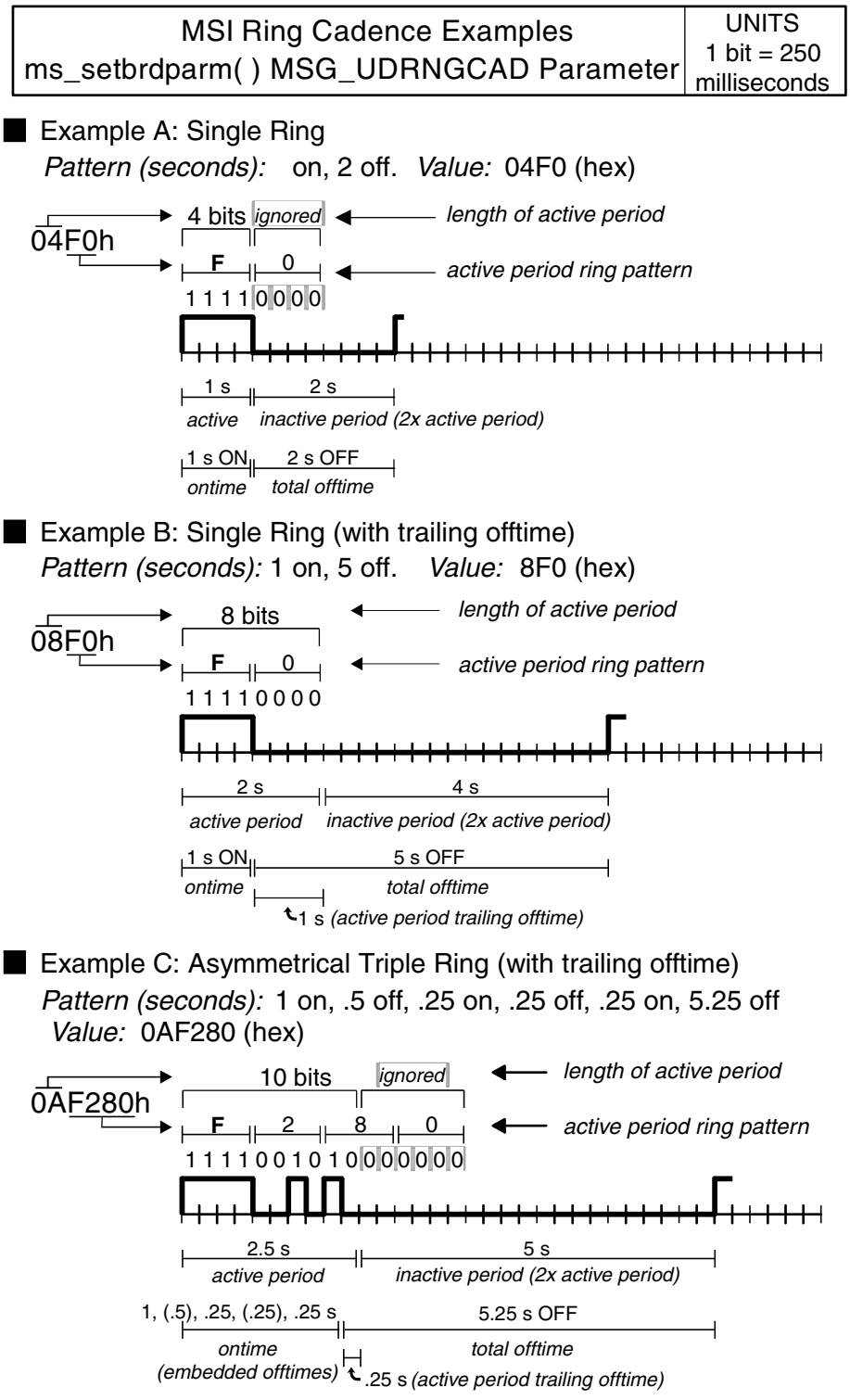
Parameter ID	Description
MSCB_ND	Defines the notify-on-add tone generated to notify conference parties that a party has joined or left the conference. **valuep** must be set to point to an MS_NCB structure that specifies tone characteristics. Note that the pulse repetition field is ignored by the function. See MS_NCB, on page 143 for structure details. **Note:** This parameter is not supported on DM3 boards.
MSCB_ZIP	Zip tone controls the characteristics of the tone generated to notify a party that they are about to be connected with a call. The volume, tone frequency, and duration fields of the MS_NCB structure are set but the pulse repetition field is ignored by the function. **Note:** This parameter is not supported on DM3 boards.

Table 3. MSI Ring Cadence Examples

	Desired Cadence (seconds)	Parameter Value (hexadecimal)		
Example Number	Ring ON Time (embedded off time)	Ring OFF Time	Total Bits (byte 1)	Active Pattern (bytes 2-n)
Single Ring Patterns:				
1	.75	7.5	0B	E000
2 [A]	1	2	04	F0
3 [2]	1	2.75	05	F0
4 [4]	1	4.25	07	F0
5 [1] [B]	1	5	08	F0
6	1.25	4.75	08	F8
7 [3]	1.5	3	06	FC
8	1.5	3.75	07	FC
9 [6]	2	4	08	FF
Double Ring Patterns:				
10 [5]	.5, (.25), .5	2.5	05	D8
11	.5, (.25), .5	4	07	D8
12	1, (.75), 1	5.5	0B	F1E0
Triple Ring Patterns:				
13	1, (.5), .25, (.25), .25	4.5	09	F280
14 [C]	1, (.5), .25, (.25), .25	5.25	0A	F280
15	1, (1), .25, (.25), .25	5.5	0B	F-A-
16	.5, (.25), .5, (.25), 1	5	0A	DBC0

[1] – [6] These show the predefined cadences for the MSG_PDRNGCAD parameter.
[A] – [C] These examples are shown in Figure 1.

Figure 1. Ring Cadence Examples



Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

main()
{
   int   devh;              /* Board device descriptor variable */
   char  cadence[7];        /* Cadence parameter array */

   if ((devh = ms_open("msiB1", 0)) == -1) {
      printf("Error opening msiB1 : errno = %d\n", errno);
      exit(1);
   }

   /*
    * Set cadence bit pattern
    * (Active cadence : 1 sec on, 0.75 secs off, 1 sec on)
    * (Inactive period : 5.5 secs off)
    */
   cadence[0] = 0x0b; /* Bit pattern 11 bits wide */
   cadence[1] = 0xf1; /* Pattern : 11110001 */
   cadence[2] = 0xeo; /* Pattern : 11100000 */

   /* Set ring cadence to the user-defined pattern */
   if (ms_setbrdparm(devh,MSG_UDRNGCAD,(void *)&cadence[0])) == -1){
      printf("Error setting board parameter : %s\n", ATDV_ERRMSGP(devh));
      exit(1);
   }

   /* Predefined selection 3 from Table 1 */
   cadence[0] = 3;

   /* Set ring-cadence to predefined pattern 3 */
   if (ms_setbrdparm(devh,MSG_PDRNGCAD,(void *)&cadence[0])) == -1){
      printf("Error setting board parameter : %s\n", ATDV_ERRMSGP(devh));
      exit(1);
   }

   if (ms_close(devh) == -1) {
      printf("Error Closing msiB1 : errno - %d\n", errno);
      exit(1);
   }
}
```

See Also

- **ms_getbrdparm( )**

ms_setcde( )

Name: int ms_setcde (devh, confID, cdt)

Inputs:

int devh	• MSI device handle
int confID	• conference identifier
MS_CDT *cdt	• pointer to an MS_CDT structure

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

The **ms_setcde( )** function changes the attributes of a party in an existing conference.

Note: If the party attributes of more than one party are to be set, this function must be called multiple times.

Parameter	Description
devh	the MSI device handle
confID	the conference identifier number
cdt	pointer to the conference descriptor table. See the MS_CDT data structure page for details.

Cautions

This function fails when:

- The device handle specified is invalid
- The device is not connected to the TDM bus

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>        /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define   NUM_PARTIES   2

int dev1;                   /* Board dev descriptor variables */
int chdev2;                 /* Channel dev descriptor */
MS_CDT cdt[NUM_PARTIES];    /* Conf. desc. table */
int confID;                 /* Conf. ID */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}

/* Open board 1, channel 2 device */
if ((chdev2 = ms_open("msiB1C2",0) == -1) {
    printf("Cannot open MSI B1, C2. errno = %d", errno);
    exit(1);
}

/*
 *
 * Continue processing
 *
 */

/* Set up CDT structure */
cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_COACH;

cdt[1].chan_num = 1;
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_PUPIL;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) != 0) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 *
 * Continue processing
 *
 */

/* Now change the attribute of MSI Station 2 */
cdt[0].chan_num = 2;
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

if((ms_setcde(dev1, confID, cdt)) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}
```

```
/*
 * Continue Processing
 *
 */
```

See Also

- **ms_addtoconf( )**
- **ms_estconf( )**
- **ms_getcde( )**

ms_setevtmsk( )

Name: ms_setevtmsk (devh, event, bitmask, action)

Inputs:

int devh	• MSI station device handle
int event	• event to be enabled/disabled
unsigned short bitmask	• bitmask for events
int action	• set, add, or subtract bitmask

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_setevtmsk( )** function changes transition event masks and enables and disables messages from a station.

Parameter	Description
devh	the valid station device handle returned by a call to **ms_open( )**
event	type of transition event to be enabled or disabled: • MSEV_SIGMSK – hook switch transition event Notification of specific signaling events is enabled or disabled by setting the **bitmask** parameter.
bitmask	the event to be enabled by setting the bitmask for that event. Multiple transition events may be enabled or disabled with one function call if the bitmasks are ORed together. The possible values for the **bitmask** parameter are: • MSMM_OFFHOOK – enables off-hook detection • MSMM_ONHOOK – enables on-hook detection • MSMM_HOOKFLASH – enables hook flash detection

Parameter	Description
action	specifies how the transition event mask is changed. Events can be added to or subtracted from those specified in **bitmask**. The possible values for the **action** parameter are: • DTA_SETMSK – enables notification of events specified in **bitmask** and disables notification of previously set events • DTA_ADDMSK – enables messages from the channel specified in bitmask, in addition to previously set events • DTA_SUBMSK – disables messages from the channel specified in bitmask

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

- Specify the events to enable in the **bitmask** field
- Specify DTA_SETMSK in the **action** field

To enable an additional event specified in **bitmask** without disabling the currently enabled events:

- Specify the events in **bitmask**
- Specify DTA_ADDMSK in the **action** field

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

- Specify the events in **bitmask**
- Specify DTA_SUBMSK in the **action** field

To disable all currently enabled events:

- Specify 0 in **bitmask**
- Specify DTA_SETMSK in the **action** field

Processing an Event

When a hook switch transition event occurs, the application receives an MSEV_SIGEVT event as the event type. The associated event data will contain the bitmask of the specific transition that caused the event. To enable an event handler for a specified event, follow these steps:

- Call **sr_enbhdlr( )**. This function specifies the event and the application defined event handler that is called from a signal handler.
- Call **ms_setevtmsk( )**. This function specifies the list of events the application should be notified of.
- For an event to be handled, it must be specified in both **sr_enbhdlr( )** and **ms_setevtmsk( )**.
- The event data is retrieved using the **sr_getevtdatap( )** function. Refer to the *Modular Station Interface API Programming Guide* for more information.

Cautions

This function fails when:

- The device handle is invalid
- The event specified is invalid

- The action specified is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

/* Basic error handler */
do_error( devh, funcname )
   int devh;
   char *funcname;
{
   int errorval = ATDV_LASTERR( devh );

   printf( "Error while calling function %s.\n", funcname );
   printf( "Error value = %d.", errorval );
   printf( "\n" );
}

main()
{
   int tsdev;       /* Channel device descriptor variable */

   /* Open board 1 time slot 1 device */
   if ( ( tsdev = ms_open( "msiB1C1", 0 ) ) == -1 ) {
      printf( "Cannot open device msiB1C1. errno = %d", errno );
      exit( 1 );
   }

   /* Enable signaling transition events (off-hook event) */
   if ( ms_setevtmsk( tsdev, MSEV_SIGMSK, MSMM_OFFHOOK, DTA_SETMSK ) == -1 ){
      do_error( tsdev, "ms_setevtmsk()" );
      exit( 1 );
   }

   /*
    *   Continue processing
    *          .
    *          .
    *          .
    */

   /* Done processing - close device  */
   if ( ms_close( tsdev ) == -1 ) {
      printf( "Cannot close board msiB1C1. errno = %d", errno );
   }
}
```

See Also

- ms_getevtmsk( )

ms_SetMsgWaitInd( )

Name: ms_SetMsgWaitInd (devh, IndicatorState, rfu1, rfu2)

Inputs:

int devh	• station device handle
unsigned short IndicatorState	• indicator ON/OFF state
void* rfu	• reserved for future use
void* rfu	• reserved for future use

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: synchronous

Platform: DI, HDSI

Description

The **ms_SetMsgWaitInd( )** function generates an FSK signal to illuminate the message waiting LED.

Parameter	Description
devh	device handle
IndicatorState	toggles FSK message waiting indicator (MWI) on a phone set • MS_MSGINDON – turns the MWI on • MS_MSGINDOFF – turns the MWI off
rfu	reserved for future use
rfu	reserved for future use

Cautions

This function can only be issued when the station is on-hook.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>    /* For Windows application only */
#include <stdio.h>
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int dev1;        /* Station Device Descriptor    */
int rc;          /* Return Code                  */

unsigned short IndicatorState= MS_MSGINDON;
void *RFU1=0,*RFU2=0;

/* Open board 1, Station 1 device */
if ( (dev1 = ms_open("msiB1C1", 0)) == -1)
{
    printf("Cannot open msiB1C1, Station 1, Channel 1: errno=%d\n",errno);
    exit(1);
}

/*
*    Continue processing
*/

/*    Set the Message Wait Indicator to ON    */
IndicatorState= MS_MSGINDON;
if((rc=ms_SetMsgWaitInd(dev1,IndicatorState,RFU1,RFU2))==-1)
{
    /* process error    */
}

/*
*    Continue processing
*/

/*    Set the Message Wait Indicator to OFF    */
IndicatorState= MS_MSGINDOFF;
if((rc=ms_SetMsgWaitInd(dev1,IndicatorState,RFU1,RFU2))==-1)
{
    /* process error    */
}

/*
*    Continue processing
*/

/* Done processing - close device */
if(ms_close(dev1)==-1)
{
printf("Cannot close device msiB1C1. errno=%d\n",errno);
    exit(1);
}
```

See Also

- **ms_genringCallerID( )**

ms_setstparm( )

Name: ms_setstparm (devh, param, valuep)

Inputs:

int devh	• MSI station device handle
unsigned char param	• parameter name
void *valuep	• pointer to parameter value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_setstparm( )** function changes the MSI station level parameters. Specifically the MSSP_STPWR parameter can be used to turn station power (loop current) on or off.

Parameter	Description
devh	the valid station device handle returned by a call to **ms_open( )**
param	specifies the station level parameter • MSSP_STPWR – station power status
valuep	specifies the address of the parameter value. Possible values are: • MS_PWROFF – power down station. Selecting this value turns off the loop current to the specified station. • MS_PWRON – power up station. Selecting this value turns on the loop current to the specified station. The station power is ON by default.

Cautions

This function fails when:

- The station device handle is invalid
- The parameter specified is invalid
- The parameter value specified is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>        /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

main()
{
   int        devh;        /* MSI/SC station device descriptor */
   int        value;       /* Parameter value */

   if ((devh = ms_open("msiB1C1", 0)) == -1) {
      printf("Error opening msiB1C1 : errno = %d\n", errno);
      exit(1);
   }

   /* Power off the station */
   value = MS_PWROFF;
   if (ms_setstparm(devh, MSSP_STPWR, (void *)&value)) == -1) {
      printf("Error setting board parameter : %s\n", ATDV_ERRMSGP(devh));
      exit(1);
   }

   if (ms_close(devh) == -1) {
      printf("Error Closing msiB1C1 : errno = %d\n", errno);
      exit(1);
   }
}
```

See Also

- **ms_getbrdparm( )**
- **ms_setbrdparm( )**

ms_setvol( )

Name: int ms_setvol (devh, type, steps)

Inputs:

int devh	• MSI station device handle
int type	• volume adjust or reset
int steps	• number of steps to increase or decrease volume

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Station

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_setvol( )** function changes or resets the station volume.

Parameter	Description
devh	the station handle
type	specifies whether to adjust or to reset current mode
steps	the number of steps to increase or decrease volume

The **type** parameter dictates whether the volume will be adjusted from its current level or reset to the default value. The **type** parameter must be set to one of the following values:

VOLADJ
Adjusts station volume

VOLRES
Resets station volume back to the default

If the **type** parameter is VOLRES, the volume is returned to the default setting of -3 dB and the third parameter, **steps**, is ignored. For VOLADJ, **steps** increases or decreases from the current volume by multiples of 1 dB. A positive **steps** value increases the volume, and a negative **steps** value decreases the volume. The volume ranges from -9 dB to +3 dB, with a default value of -3 dB. Hence, the volume can be changed 6 dB higher or lower from the default value. However, depending on the current volume setting, the number of steps in either direction will be limited.

Note: An error will **not** be returned if the saturation point is reached in either direction.

Cautions

This function fails when:

- An invalid device handle is specified
- The device is not connected to the MSI board

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

int  chdev2;                 /* Station dev descriptor */

/* Open board 1, station 2 device */
if ((chdev2 = ms_open("msiB1C2",0) == -1) {
   printf("Cannot open MSI B1, C2. errno = %d", errno);
   exit(1);
}

/*
 *
 * Continue processing
 *
 */

/* Increase volume by 2 dB from current level */
if (ms_setvol(chdev2,VOLADJ,2)==-1) {
   printf("Error setting volume: %s", ATDV_ERRMSGP(chdev2));
   exit(1);
}

/*
 * Continue Processing
 *
 */
```

See Also

None.

ms_stopfn( )

Name: int ms_stopfn (devh, funcid)

Inputs: unsigned int devh • MSI station device handle

unsigned int funcid • ID of multitasking function

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Device Management

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_stopfn( )** function stops a multitasking function in progress for a station. Currently, ringing is the only type of multitasking function that can be stopped. However, ringing may be started by one of the three following functions: **ms_genring( )**, **ms_genringCallerID( )**, and **ms_genringex( )**.

Parameter	Description
devh	the MSI station device handle
funcid	the identification of the multitasking function that must be stopped. The valid value is: • MTF_RING – Stops ringing on a station, if in progress.

Cautions

This function fails when:

- The device specified is not an MSI station
- The parameter specified is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

 int chdev1 ;

/* Open board 1, station 2 device */
if ((chdev1 = ms_open("msiB1C2",0) == -1) {
    printf("Cannot open MSI B1, C2. errno = %d", errno);
    exit(1);
}

/* ring the station 2 five times */
if (ms_genring(chdev1, 5, EV_ASYNC)== -1){
    printf("Error Message = %s",ATDV_ERRMSGP(chdev1));
    exit(1);
}

/* 2 seconds later, ringing has not completed and station 2
 * has not gone off-hook. However, there is a need to abort the
 * ringing on station 2. Issue the abort command
 */

if (ms_stopfn(chdev1,MTF_RING)== -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(chdev1));
    exit(1);
}
```

See Also

- **ms_genring( )**
- **ms_genringCallerID( )**
- **ms_genringex( )**

ms_tstcom( )

Name: int ms_tstcom (devh, tmo)

Inputs:
- int devh — • MSI device handle
- int tmo — • time-out value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Diagnostic

Mode: asynchronous or synchronous

Platform: DI, HDSI, Springware

Description

The **ms_tstcom( )** function tests the ability of a board to communicate with the system. This function can operate in either blocking or non-blocking mode.

Parameter	Description
devh	the valid board device handle returned by a call to **ms_open( )**
tmo	the maximum amount of time that the function will block while waiting for a response from the board. If a response is not returned within **tmo** seconds, an error will be returned.

To run this function in synchronous (blocking) mode, set **tmo** to the length of time, in seconds, to await a return. If a response is not returned within **tmo** seconds, an error is returned.

To operate this function in asynchronous (non-blocking) mode, specify 0 for **tmo**. This allows the application to continue processing while awaiting a completion event. If event handling is properly set up for your application, DTEV_COMRSP will be returned by the **sr_getevttype( )** function included in the SRL when the test completes successfully. See the *Modular Station Interface API Programming Guide* for information about event handling.

Cautions

- This is a board-level function only.
- This function fails when:
  - The device handle is invalid
  - There is a hardware problem on the board
  - There is a configuration problem (IRQ conflict)

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

/* Basic error handler */
do_error( devh, funcname )
   int devh;
   char *funcname;
{
   int errorval = ATDV_LASTERR( devh );
   printf( "Error while calling function %s.\n", funcname );
   printf( "Error value = %d.", errorval );
   printf( "\n" );
}

main()
{
   int bddev;          /* Board device descriptor variable */

   /* Open board 1 device */
   if ( ( bddev = ms_open( "msiB1", 0 ) ) == -1 ) {
      printf( "Cannot open board msiB1. errno = %d", errno );
      exit( 1 );
   }

   /*
    * Test the board's ability to communicate with the system.
    */
   if ( ms_tstcom( bddev, 60 ) == -1 ) {
      do_error( bddev, "ms_tstcom()" );
      exit( 1 );
   }

   printf("Communications test completed successfully\n");

   /*
    *   Continue processing
    */

   /* Done processing - close device */
   if ( ms_close( bddev ) == -1 ) {
      printf( "Cannot close board msiB1. errno = %d", errno );
   }
}
```

See Also

- **ms_tstdat( )**

ms_tstdat( )

Name: int ms_tstdat (devh, tmo)

Inputs:
- int devh — • MSI device handle
- int tmo — • time-out value

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Diagnostic

Mode: asynchronous or synchronous

Platform: DI, HDSI, Springware

Description

The **ms_tstdat( )** function performs a data test on the MSI board and verifies the integrity of the MSI interface to the PC. The data test is performed by sending a series of bytes to the MSI and by checking the integrity of the bytes returned. The function can operate synchronously or asynchronously.

Parameter	Description
devh	the valid board device handle returned by a call to **ms_open( )**
tmo	the maximum amount of time that the function will block while waiting for a response from the board. If a response is not returned within **tmo** seconds, an error will be returned.

To run this function in synchronous (blocking) mode, set **tmo** to the length of time (in seconds) to await a return. If a response is not returned within **tmo** seconds, an error is returned.

To operate this function in asynchronous (non-blocking) mode, specify 0 for **tmo**. This allows the application to continue processing while awaiting a completion event. If event handling is properly set up for your application, DTEV_DATRSP will be returned by the **sr_getevttype( )** function included in the SRL when the test completes successfully. See the *Modular Station Interface API Programming Guide* for information about event handling.

Cautions

- This is a board-level function only.
- This function fails when:
  - The test data is corrupted
  - The device handle is invalid

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

The following code example demonstrates the use of **ms_tstdat( )** in the synchronous mode.

```
#include <windows.h>         /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

/* Basic error handler */
do_error( devh, funcname )
   int devh;
   char *funcname;
{
   int errorval = ATDV_LASTERR( devh );
   printf( "Error while calling function %s.\n", funcname );
   printf( "Error value = %d.", errorval );
   printf( "\n" );
}

main()
{
   int bddev;         /* Board device descriptor variable */

   /* Open board 1 device */
   if ( ( bddev = ms_open( "msiB1", 0 ) ) == -1 ) {
      printf( "Cannot open board msiB1. errno = %d", errno );
      exit( 1 );
   }

   /* Perform a data integrity test between the board and PC. */
   if ( ms_tstdat( bddev, 60 ) == -1 ) {
      do_error( bddev, "ms_tstdat()" );
      exit( 1 );
   }

   printf("Data integrity test completed successfully\n");

/*
    *   Continue processing
    *          .
    *          .
    *          .
    */

   /* Done processing - close device */
   if ( ms_close( bddev ) == -1 ) {
      printf( "Cannot close board msiB1. errno = %d", errno );
   }
}
```

See Also

- ms_tstcom( )

ms_unlisten( )

Name: int ms_unlisten (devh)

Inputs: int devh
- MSI station device handle

Returns: 0 on success
-1 on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: TDM Bus Routing

Mode: synchronous

Platform: DI, HDSI, Springware

Description

The **ms_unlisten( )** function disconnects the listen (receive) component of a station interface from a TDM bus time slot.

Parameter	Description
devh	the station device handle

Cautions

This function fails when an invalid station device handle is specified.

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>          /* For Windows applications only */
#include  "srllib.h"
#include  "dxxxlib.h"
#include  "dtilib.h"

int       chdev               /* Station device handle */
```

```
/* Open board 1, channel 1 */
if ((chdev = ms_open("msiB1C1",0)) == -1) {
   printf("Cannot open channel msiB1C1. errno = %d", errno);
   exit(1);
}

/* Disconnect receive of board 1, station 1 from all SCbus time slots */
if (ms_unlisten(chdev) == -1) {
   printf("Error message = %s",ATDV_ERRMSGP(chdev));
   exit(1);
}
```

See Also

- ms_listen( )

ms_unmonconf( )

Name: int ms_unmonconf (devh, confID)

Inputs:

int devh	• MSI board device handle
int confID	• conference ID

Returns: 0 on success
-1 on failure (if the board is **not** an MSI board)

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

Platform: DI, Springware

Description

Note: This function is not supported on Intel® NetStructure™ High Density Station Interface (HDSI) boards, however, it is supported on Intel® Dialogic® Integrated Series boards.

The **ms_unmonconf( )** function removes a monitor from a conference. Calling this function frees one resource.

Parameter	Description
devh	the MSI board device handle
confID	the conference identifier

Cautions

This function fails when:

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

Errors

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

For information about error codes, refer to Chapter 5, "Error Codes". Error defines can be found in *dtilib.h* or *msilib.h*.

Example

```
#include <windows.h>           /* For Windows applications only */
#include <errno.h>
#include "srllib.h"
#include "dtilib.h"
#include "msilib.h"

#define  NUM_PARTIES  2

int  dev1;                     /* Board dev descriptor variables */
int  tsdev1;                   /* DTI time slot device handle */
MS_CDT cdt[NUM_PARTIES];       /* conference descriptor table */
int  confID;                   /* conference ID */
long  lts;                     /* listen time slot */
SC_TSINFO       tsinfo;        /* time slot information structure */

/* Open board 1 device */
if ((dev1 = ms_open("msiB1",0)) == -1) {
    printf( "Cannot open MSI B1: errno=%d", errno);
    exit(1);
}

/* Assume that there is a DTI in the system.
 * Assume the device handle for a time slot on the DTI
 * is tsdev1 and time slot it is assigned to is ts1
*/

/* Set up CDT structure */
cdt[0].chan_num = station ;     /* Valid MSI Station */
cdt[0].chan_sel = MSPN_STATION;
cdt[0].chan_attr = MSPA_NULL;

cdt[1].chan_num = ts1 ;      /*  ts1 is the DTI time slot */
cdt[1].chan_sel = MSPN_TS;
cdt[1].chan_attr = MSPA_RO;

/* Establish conference */
if (ms_estconf(dev1, cdt, NUM_PARTIES, MSCA_ND, &confID) == -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/*
 * Continue Processing
 */

/* Now monitor the conference */
if (ms_monconf(devh,confID,&lts)== -1) {
    printf("Error Message = %s",ATDV_ERRMSGP(dev1));
    exit(1);
}

/* Assume that a DTI time slot  tsdev1 is the available  */
tsinfo.sc_numts = 1;
tsinfo.sc_tsarrayp = &lts;

if (dt_listen(tsdev1,&tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
    exit(1);
}
/*
 * Continue processing
 */
```

```
/* Unlisten to the monitor's time slot first */
if (dt_unlisten(tsdev1) == -1) {
    printf("Error message = %s\n", ATDV_ERRMSGP(tsdev1);
    exit(1);
}

/* Now unmonitor the conference */
if (ms_unmonconf(devh, confID) == -1) {
    printf("Error message = %s\n", ATDV_ERRMSGP(devh));
    exit(1);
}

/* Continue processing */
```

See Also

- ms_estconf( )
- ms_monconf( )

Events 3

This chapter contains an alphabetical list of the events returned by the modular station interface (MSI) software functions.

MSEV_DATASENT
: Termination event. Indicates data was successfully sent by **ms_SendData( )** when the function is called in asynchronous mode.

MSEV_ERREVT
: Error event.

MSEV_NORING
: Termination event. Indicates failure of **ms_genring( )**, **ms_genringCallerID( )**, or **ms_genringex( )** in asynchronous mode.

MSEV_RING
: Termination event. Indicates successful completion of **ms_genring( )**, **ms_genringCallerID( )**, or **ms_genringex( )** in asynchronous mode. The event data for MSEV_RING is:
  - MSMM_RNGOFFHK – Solicited off-hook was detected.
  - MSMM_RNGSTOP – Ringing was stopped by ms_stopfn( ).
  - MSMM_TERM – Ringing was terminated.

MSEV_SENDDATAFAILED
: Termination event. Indicates failure of send data operation when **ms_SendData( )** is called in asynchronous mode.

MSEV_SIGEVT
: Signaling transition event. The event data for MSEV_SIGEVT is:
  - MSMM_HOOKFLASH – Line device detected hook flash.
  - MSMM_OFFHOOK – Line device has gone off-hook.
  - MSMM_ONHOOK – Line device has gone on-hook.

Data Structures 4

This chapter alphabetically lists the data structures used by the modular station interface (MSI) library functions. These structures are used to control the operation of functions and to return information. In this chapter, the data structure definition is followed by a detailed description of the fields in the data structure. The fields are listed in the sequence in which they are defined in the data structure.

- CT_DEVINFO. 134
- MS_CADENCE. 137
- MS_CDT 139
- MS_DataInfo 141
- MS_NCB 143
- SC_TSINFO 144

CT_DEVINFO

```
typedef struct ct_devinfo {
   unsigned long  ct_prodid;       /* product ID */
   unsigned char  ct_devfamily;    /* device family */
   unsigned char  ct_devmode;      /* device mode */
   unsigned char  ct_nettype;      /* network interface */
   unsigned char  ct_busmode;      /* bus architecture */
   unsigned char  ct_busencoding;  /* bus encoding */
   union {
           unsigned char ct_RFU[7];     /* reserved */
           struct {
                 unsigned char ct_prottype;
           } ct_net_devinfo;
   } ct_ext_devinfo;
} CT_DEVINFO;
```

Description

This is a channel/station information structure for MSI boards.

Valid values for each member of the structure are defined in *ctinfo.h*, which is referenced by *dtilib.h*.

Field Descriptions

On **DM3 boards**, the fields of the CT_DEVINFO data structure are described as follows:

ct_prodid
: Contains a valid product identification number for the device [length: 4 (unsigned long)].

ct_devfamily
: Specifies the device family [length: 1 (unsigned char)]. Possible values are:
  - CT_DFDM3 – DM3 device
  - CT_DFHMPDM3 – HMP device (Host Media Processing)

ct_devmode
: Specifies the device mode [length: 1 (unsigned char)] that is valid only for a D/xx or VFX/xx board. Possible values are:
  - CT_DMRESOURCE – DM3 voice device in flexible routing configuration
  - CT_DMNETWORK – DM3 network device or DM3 voice device in fixed routing configuration

  For information about flexible routing and fixed routing, see the *Voice API Programming Guide*.

ct_nettype
: Specifies the type of network interface for the device [length: 1 (unsigned char)]. Possible values are:
  - CT_IPT – IP connectivity
  - CT_NTANALOG – analog interface. Analog and voice devices on board are handling call processing
  - CT_NTT1 – T1 digital network interface
  - CT_NTE1 – E1 digital network interface
  - CT_NTMSI – MSI/SC station interface

- CT_NTHIZ – high impedance (HiZ) interface. This value is bitwise-ORed with the type of network interface. A digital HiZ T1 board would return CT_NTHIZ | CT_NTT1. A digital HiZ E1 board would return CT_NTHIZ | CT_NTE1. An analog HiZ board would return CT_NTHIZ | CT_NTTXZSWITCHABLE | CT_NTANALOG.
- CT_NTTXZSWITCHABLE – The network interface can be switched to the transmit impedance state. This value is bitwise-ORed with the type of network interface. An analog HiZ board would return CT_NTHIZ | CT_NTTXZSWITCHABLE | CT_NTANALOG. This is used to transmit the record notification beep tone.

ct_busmode

Specifies the bus architecture used to communicate with other devices in the system [length: 1 (unsigned char)]. Possible values are:

- CT_BMSCBUS – TDM bus architecture
- CT_H100 – H.100 bus
- CT_H110 – H.110 bus

ct_busencoding

Describes the PCM encoding used on the bus [length: 1 (unsigned char)]. Possible values are:

- CT_BEULAW – mu-law encoding
- CT_BEALAW – A-law encoding
- CT_BELLAW – linear encoding
- CT_BEBYPASS – encoding is being bypassed

ct_rfu

Returned by **ms_getctinfo( )** for DM3 MSI devices. This field returns a character string containing the board and channel of the voice channel resource associated with the station interface. This data is returned in BxxCy format, where xx is the voice board and y is the voice channel. For example, dxxxB1C1 would be returned as B1C1. To subsequently use this information in a dx_open( ) function, you must add the dxxx prefix to the returned character string.

ct_ext_devinfo.ct_net_devinfo.ct_prottype

Contains information about the protocol used on the specified digital network interface device. Possible values are:

- CT_CAS – channel associated signaling
- CT_CLEAR – clear channel signaling
- CT_ISDN – ISDN
- CT_R2MF – R2MF

On **Springware boards**, the fields of the CT_DEVINFO data structure are described as follows:

ct_prodid

Contains a valid product identification number for the device [length: 4 (unsigned long)].

ct_devfamily

Specifies the device family [length: 1 (unsigned char)]. Possible values are:

- CT_DFD41D – D/41D board family
- CT_DFD41E – analog or voice channel of a D/xx or VFX/xx board such as D/41ESC or VFX/40ESC
- CT_DFSPAN – analog channel such as of a D/160SC-LS board; a voice channel such as of a D/240SC, D/320SC, D/240SC-T1, D/300SC-E1, or D/160SC-LS board; or a digital channel such as of a D/240SC-T1 or D/300SC-E1 board
- CT_DFMSI – a station on an MSI board

- CT_DFSCX – SCX160 SCxbus adapter family

ct_devmode
Specifies the device mode field [length: 1 (unsigned char)] that is valid only for a D/xx or VFX/xx board. Possible values are:
- CT_DMRESOURCE – analog channel not in use
- CT_DMNETWORK – analog channel available to process calls from the telephone network

ct_nettype
Specifies the type of network interface for the device [length: 1 (unsigned char)]. Possible values are:
- CT_NTNONE – D/xx or VFX/xx board configured as a resource device; voice channels are available for call processing; analog channels are disabled.
- CT_NTANALOG – analog and voice devices on board are handling call processing
- CT_NTT1 – T1 digital network interface
- CT_NTE1 – E1 digital network interface
- CT_NTMSI – MSI/SC station interface

ct_busmode
Specifies the bus architecture used to communicate with other devices in the system [length: 1 (unsigned char)]. Possible values are:
- CT_BMSCBUS – TDM bus architecture

ct_busencoding
Describes the PCM encoding used on the bus [length: 1 (unsigned char)]. Possible values are:
- CT_BEULAW – mu-law encoding
- CT_BEALAW – A-law encoding

ct_rfu
Reserved for future use.

ct_ext_devinfo.ct_net_devinfo.ct_prottype
Contains information about the protocol used on the specified digital network interface device. Possible values are:
- CT_CAS – channel associated signaling
- CT_CLEAR – clear channel signaling
- CT_ISDN – ISDN
- CT_R2MF – R2/MF signaling

MS_CADENCE

```
typedef struct ms_cadence {
   BYTE cadid;       // Cadence ID, 1-MS_MAX_CADID
   BYTE cadlength;   // Cadence Length
   BYTE *cadpattern; // Pointer to Cadence Pattern
} MS_CADENCE;
```

Description

MS_CADENCE is used by the **ms_getbrdparm( )** and **ms_setbrdparm( )** functions when manipulating the **MSG_DISTINCTRNG** parameter**.** The structure contains cadence setting information used for initializing a distinctive ring cadence.

Notes: ***1.*** Distinctive ring and board-level ring cadence are mutually exclusive except in the case where the cadence lengths are identical. You will get an E_MSRNGCADCNFLCT error if the MSG_PDRNGCAD or MSG_UDRNGCAD board-level ring cadence is currently set to a cadence that does not match the distinctive ring cadence length. For example, if MSG_UDRNGCAD is set to a cadence length of 4, you cannot initialize distinctive ring Group A, which uses a length of 6 seconds.

2. You cannot remove, change, or overwrite the distinctive ring definition once a pattern has been assigned to a cadence ID.

For MSI boards only: when a ring cycle shorter than the default cycle of 6 seconds (2 seconds ON, 4 seconds OFF) is desired, the *RING.PRM* parameter file must be downloaded and edited to define a new default active cycle length. A valid range of active cycle lengths is from 1 - 6 seconds. For more information, see the DCM on-line help.

Field Descriptions

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

cadid
: Assigns a cadence ID to the pattern. Range: 1 - 8. Returns an E_MSBADRNGCAD error if outside the range.

cadlength
: This must be set to MS_RNGA_CADLEN, which is a cadence length of 6-seconds, to initialize a cadence pattern from Table 4, “Ring Cadence Group A”, on page 138.

cadpattern
: The user may specify any cadence pattern, subject to the same restrictions as detailed under MSG_UDRNGCAD. However, if the board-wide cadence length has the default value of 6 seconds, the user may specify one of the predefined patterns in Table 4, or a cadence pattern may be defined by the user.

Table 4. Ring Cadence Group A

Cadence Pattern Name	Ring Cadence Pattern (in seconds)
MS_RNGA_TWOSEC	2 on, 4 off
MS_RNGA_ONESEC	1 on, 5 off
MS_RNGA_SPLASH1	.5 on, 5.5 off
MS_RNGA_SPLASH2	.5 on, .25 off, .5 on, 4.75 off
MS_RNGA_SPLASH3	.5 on, .25 off, .5 on, .25 off, .5 on, 4 off
MS_RNGA_SPLASH4	.25 on, .25 off, .25 on, .25 off, .25 on, .25 off, .25 on, 4.25 off
MS_RNGA_LONGSHORT	1.25 on, .25 off, .5 on, 4 off
MS_RNGA_SHORTLONG	.5 on, .25 off, 1.25 on, 4 off
Notes: MS_RNGA_SPLASH3 and MS_RNGA_SPLASH4 are not supported on DM3 boards. The splash ring cycles apply to MSI/SC boards. They do not apply to the PCI version.	

MS_CDT

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

Description

The conference descriptor table is an array of MS_CDT structures.

An extended connection can also be described by the descriptor table. For an extended connection, there are two entries in the table. The order of the entries in the table is significant. The first entry must be the connection identifier, the second must be the connection extender.

Field Descriptions

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

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

chan_sel
: defines the meaning of chan_num. Valid choices are as follows:
  - MSPN_STATION – MSI station number
  - MSPN_TS – TDM bus time slot number

chan_attr
: bitmask describing the party's properties within the conference. Valid choices are:
  - MSPA_NULL – No special attributes for party (default)
  - MSPA_RO – Party participates in conference in receive-only mode
  - MSPA_TARIFF – Party receives periodic tone for duration of call
  - MSPA_DIG – Digital front end (applicable to Springware boards only)
  - MSPA_COACH – Party is a coach. Coach heard by pupil only.
  - MSPA_PUPIL – Party is a pupil. Pupil hears everyone including coach.
  - MSPA_NOAGC – Disables automatic gain control (AGC)

  The following values are applicable to DM3 and DI/SI boards only:
  - MSPA_ECHOXCLEN – Echo cancel enable
  - MSPA_BROADCASTEN – Broadcast enable
  - MSPA_MODENULL – Null party
  - MSPA_MODERECVONLY – Receive only party
  - MSPA_MODEXMITONLY – Transmit only party
  - MSPA_MODEFULLDUPLX – Full duplex (same as a party with no specific attributes in Springware)
  - MSPA_PARTYTONECLAMP – Tone clamping per conferee

  Note: If the first party (connection identifier) is in a pupil-coach situation, the party must be defined with the MSPA_PUPIL attribute when the extended connection is established. There is no way of changing the attribute of the first party once an extended connection has been established.

Table 5 shows the allowable combinations of attributes within a conference, where each row represents an allowable combination.

Table 5. Valid Attribute Combinations

Row No.	AGC Disabled	Pupil	Coach	Periodic Tone	Receive-only mode
1					X
2				X	
3				X	X
4			X		
5		X			
6		X		X	
7	X	X			
8	X	X		X	

Notes: ***1.*** Only one coach and one pupil are allowed in a conference at any time.

2. The default MSPA_NULL must be used if channel attributes are not set.

3. The MSPA_NOAGC option should only be used when the connection identifier is a pupil. This ensures that the client will not hear a change in the pupil's volume when the connection is extended.

4. If the coach speaks before any conversation has taken place between the client and the pupil, the client will hear some background noise for a fraction of a second. Under most circumstances, this will not be an issue since the coach generally does not need to speak before some conversation has taken place between the client and the pupil.

MS_DataInfo

```
typedef struct ms_DataInfo
{
     unsigned int    version;
     eMSSendDataType dataType;    /* Data Type - FSK  */
     union
     {
          char* dataString;
     }uInfo;
} MS_DataInfo, *MS_DataInfoPtr;
```

Description

This structure is used by the **ms_SendData( )** function. It contains call information and is used to transmit data about an incoming call to a station that is already in a call. This operation is commonly called call waiting caller ID.

Field Descriptions

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

version
reserved for future use. Set to 0.

dataType
type of data to be sent. Values include:

- eMSInvalidDataType – RFU
- eMSFSK – FSK (frequency shift keyed) data
- eMSMaxDataType – RFU

DataString
ASCII character string that holds information about the origination party. The maximum length is 127 characters. The following sub-fields are supported when sending an FSK caller ID string:

- Caller Name – identifies the name of the call originator if available.
- Caller Name Absence Reason – identifies why call originator's name is not available. Possible reasons are Private (P) or Out of Area (O).
- Caller Number – identifies the number of the call originator if available.
- Caller Number Absence Reason – identifies why call originator's number is not available. Possible reasons are Private (P) or Out of Area (O).
- Date Time – identifies the date and time at which the call is sent.

Sub-group identifiers with format **X:** are used to specify sub-fields for caller ID transmission. Note that the user strings embed this character as part of sub-field identifiers. Thus this sub-group identifier is implicit by nature.

- A: – Identifies beginning of Caller Number Absence Reason sub-field.
- B: – Identifies beginning of Caller Name Absence Reason sub-field.
- I: – Identifies beginning of Caller Number sub-field.
- N: – Identifies beginning of Caller Name sub-field.
- T: – Identifies beginning of Time and Date sub-field.

Notes: ***1.*** Use the character '/' as an escape character to indicate that ':' is part of the string. For example, Next string "N:J/:NamathI:993-3000" uses the escape character / to embed the name J:Namath.

2. The end of a sub-field is recognized by the character ":" or the end of string when a sub-field is located at the end of the string.

MS_NCB

```
typedef struct ms_ncb{
   unsigned char volume;      /* volume */
   unsigned char tone;        /* tone frequency */
   short duration;            /* tone duration */
   short pulse;               /* pulse repetition
                                 interval */
} MS_NCB
```

Description

The MS_NCB structure is used by the **ms_setbrdparm( )** function when setting certain parameters. The structure contains tone information for the **MSCB_ND** parameter (notify-on-add tone) and **MSCB_ZIP** parameter (zip tone).

Field Descriptions

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

volume
: tone volume

: default for **MSCB_ND** and **MSCB_ZIP** = 7

tone
: frequency of tone

: default for **MSCB_ND** = 0x24H (1125 Hz); default for **MSCB_ZIP** = 0x18H

duration
: duration of tone

: default for **MSCB_ND** = 0x14H (200 ms); default for **MSCB_ZIP** = 0x64H (1 sec)

pulse
: reserved for future use

SC_TSINFO

```
typedef struct {
      unsigned long     sc_numts;
      long             *sc_tsarrayp;
} SC_TSINFO;
```

Description

This structure defines the TDM bus time slot information. It is used by **ms_getxmitslot( )** and **ms_listen( )**.

Field Descriptions

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

sc_numts
: specifies the total number of TDM bus time slots to which the connection is to be made

sc_tsarrayp
: pointer to an array which contains the TDM bus time slots (between 0 and 1023) to be connected to the receive of the station device

Error Codes 5

This chapter describes the error/cause codes supported by the modular station interface (MSI) library, *msilib.h*.

All MSI library functions return a value that indicates the success or failure of the function call. Success is indicated by a return value of zero or a non-negative number. Failure is indicated by a value of -1. If a function fails, call the Standard Attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** for the reason for failure. These functions are described in the *Standard Runtime Library API Library Reference*.

The MSI library contains the following error codes, listed in alphabetical order.

E_MS1PTY
: Cannot remove party from one-party conference

E_MSBADCHPARM
: Invalid channel parameter number

E_MSBADRNGCAD
: Invalid ring cadence identifier

E_MSBADRNGSTA
: Cannot ring station - station already off-hook

E_MSBADVAL
: Invalid parameter value

E_MSCHASNCNF
: Channel is assigned to conference

E_MSCNFFUL
: Conference system is full

E_MSCNFLMT
: Exceeds conference limit

E_MSCNTXTD
: Station is in extended connection

E_MSERRCHANSTATE
: Error returned while setting a channel state; this error also received for repeated PWRON or PWROF set chan state

E_MSGLOBREAD
: Cannot read parameter globally

E_MSINVCATTR
: Invalid conference attribute selector

E_MSINVCB
: Invalid control block ID

E_MSINVCNF
Invalid conference number

E_MSINVDATATYPE
Invalid data type specified when sending data to the station

E_MSINVDSP
Invalid DSP specified

E_MSINVFEMID
Invalid identifier read from FEM ID PAL

E_MSINVMT
Invalid multitasking function

E_MSINVPATTR
Invalid party attribute

E_MSINVPEB
Invalid PEB rate for present clock rate

E_MSINVPTYCNT
Invalid number of parties specified

E_MSINVPTYNUM
Invalid party number specified

E_MSINVPTYTYPE
Invalid conference member type

E_MSINVRNGCNT
Invalid number of ring counts

E_MSINVST
Invalid station

E_MSINVTS
Invalid time slot number

E_MSINVVAL
Bad global parameter value

E_MSINVVERSION
Invalid version number specified

E_MSINVXTD
Invalid extended connection number

E_MSINVXTDM
Invalid extended connection member

E_MSMONEXT
Monitor already exists for this conference

E_MSNOCNF
No conferencing available on device

E_MSNOCNT
Station not connected

E_MSNODSPTS
All time slots going to the DSP are busy

E_MSNOFEMCH
No MSI daughterboard to support this channel

E_MSNOMON
No monitor exists for this conference

E_MSNONCNFCH
Channel not assigned to specified conference

E_MSNONRNGBRD
Error ringing a non-ringing board

E_MSNOTS
No time slot assigned to channel

E_MSNOTSALLOC
No time slots allocated to the board

E_MSPTYASN
Party already assigned

E_MSRNGCADCNFLCT
Conflict between board-level and distinctive ring cadence lengths

E_MSSNDZIP
Sending a zip tone to this station

E_MSSTASN
Time slot already assigned to station

E_MSSYSTEM
System error- see **errno** for actual error

E_MSTSASN
Time slot already assigned to a station

E_MSTSASNCNF
Time slot already assigned to a conference

E_MSTSNOTEQ
Time slots not equal for zip tones

E_MSZIPEN
Zip tones disabled - message not allowed

E_MSZIPON
Station is currently “zipping”

Glossary

ACD: Automatic call distributor. An automated (usually software-driven) system that connects incoming calls to agents based on a distribution algorithm. The system also gathers traffic analysis statistics, such as number of calls per hour, average time holding, and call length.

agent: An operator, transcriber, telemarketing or sales representative, or other employee. In this guide, agent refers to any person using an analog station device who can be connected to a caller or recorded message through the MSI board.

A-Law: A pulse code modulation (PCM) algorithm used in digitizing telephone audio signals in E1 areas.

analog: In this guide, analog refers to agent communications between a headset and the MSI or to the loop-start type of network interface.

asynchronous function: Allows program execution to continue without waiting for a task to complete. Contrast with *synchronous function*.

automatic call distributor: See *ACD*.

baseboard: A term used in voice processing to mean a printed circuit board without any daughterboards attached.

blocking mode: When a telephone call cannot be completed, it is said that the call is "blocked". In blocking mode, it is said that the caller is "receiving a busy".

channel: 1. When used in reference to an Intel® Dialogic® digital expansion board, a data path, or the activity happening on that data path. 2. When used in reference to the CEPT telephony standard, one of 32 digital data streams (30 voice, 1 framing, 1 signaling) carried on the 2.048 MHz/sec E1 frame. (See *time slot*.) 3. When used in reference to a bus, an electrical circuit carrying control information and data.

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: C programming term for a data element consisting of fields, where each field may have a different type definition and length. The elements of a data structure usually share a common purpose or functionality, rather than being similar in size, type, etc.

daughterboard: In the context of this guide, the MSI daughterboard assembly. The daughterboard enables the MSI hardware to interface to analog station devices.

device: Any computer peripheral or component that is controlled through a software device driver.

digital: Information represented as binary code.

DIP switch: A switch usually attached to a printed circuit board with two settings: on or off. DIP switches are used to configure the board in a semi-permanent way.

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 that provides a defined interface between a program and the hardware.

DTMF: Dual Tone Multi-Frequency. DTMF refers to the combination of two tones which represents a number on a telephone key pad. Each push button has its own unique combination of tones.

E1: Another name given to the CEPT digital telephony format devised by the CCITT that carries data at the rate of 2.048 Mbps (DS-1 level). This service is available in Europe and some parts of Asia.

event: An unsolicited communication from a hardware device to an operating system, application, or driver. Events are generally attention-getting messages, allowing a process to know when a task is complete or when an external event occurs.

Extended Attribute functions: Class of functions that take one input parameter (a valid device handle) and return device-specific information.

flash: A signal generated by a momentary on-hook condition. This signal is used by the voice hardware to alert a telephone switch that special instructions will follow. It usually initiates a call transfer. See also *hook state*.

frequency shift keying (FSK): A frequency modulation technique used to send digital data over voice band telephone lines.

full-duplex: Transmission in two directions simultaneously, or more technically, bi-directional, simultaneous two-way communications.

hook flash: See *flash*.

hook state: A general term for the current line status of the channel: either on-hook or off-hook. A telephone station is said to be on-hook when the conductor loop between the station and the switch is open and no current is flowing. When the loop is closed and current is flowing, the station is off-hook. These terms are derived from the position of the old fashioned telephone set receiver in relation to the mounting hook provided for it.

host PC: The system PC in which Intel® Dialogic® hardware and software are installed and applications are run and/or developed.

IRQ: Interrupt request. A signal sent to the central processing unit (CPU) to temporarily suspend normal processing and transfer control to an interrupt handling routine. Interrupts may be generated by conditions such as completion of an I/O process, detection of hardware failure, power failures, etc.

loop start interfaces: Devices, such as an analog telephones, that receive an analog electric current. For example, taking the receiver off hook closes the current loop and initiates the calling process.

Mu-Law: The PCM coding and companding standard used in Japan and North America (T1 areas).

MSI/SC: Modular Station Interface. An SCbus-based expansion board that interfaces SCbus time slots to analog station devices.

off-hook: The state of a telephone station when the conductor loop between the station and the switch is closed and current is flowing. When a telephone handset is lifted from its cradle (or an equivalent condition occurs), the telephone line state is said to be off-hook. See also *hook state*.

on-hook: Condition or state of a telephone line when a handset on the line is returned to its cradle (or an equivalent condition occurs). See also *hook state*.

PCM: Pulse Code Modulation. The most common method of encoding an analog voice signal into a digital bit stream. PCM refers to one technique of digitization. It does not refer to a universally accepted standard of digitizing voice.

rfu: Reserved for future use.

SCbus (Signal Computing Bus): A hard-wired connection between switch handlers on SCbus-based products. SCbus is a third generation TDM (time division multiplexed) resource sharing bus that allows information to be transmitted and received among resources over 1024 time slots. See *TDM Bus*.

SCSA: Signal Computing System Architecture. A generalized open-standard architecture describing the components and specifying the interfaces for a signal processing system for the PC-based voice processing, call processing, and telecom switching industry.

Signal Computing System Architecture: See *SCSA*.

Springware: Software algorithms built into the downloadable firmware that provides the voice processing features available on all Intel® Dialogic® 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: Standard Runtime Library containing Event Management functions, Standard Attribute functions, and data structures that are used by all Intel® NetStructure™ and Intel® Dialogic® devices.

Standard Attribute functions: Class of functions that take one input parameter (a valid device handle) and return generic information about the device. The SRL contains Standard Attribute functions for all Intel® NetStructure™ and Intel® Dialogic® devices. Standard Attribute function names are case-sensitive and must be in capital letters. See *Extended Attribute functions*.

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

T1: The digital telephony format used in North America and Japan that carries data at the rate of 1.544 Mbps (DS-1 level).

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.

TDM bus routing functions: Used to set up communications between devices connected to the TDM bus. These functions enable an application to connect or disconnect (make or break) the receive (listen) channel of a device to or from a TDM bus time slot.

time slot: 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.

zip tone: Short burst of a specified tone to an ACD agent headset usually indicating a call is being connected to the agent console.

Index

A

ATMS_STATINFO(_) 18
ATMS_TSSGBIT(_) 20
attribute functions 11
    ms_dsprescount(_) 35
    ms_getctinfo(_) 69

C

cadence information structure 137
cadence, distinctive ringing 54
call waiting caller ID information structure 141
channel/station information structure 134
conference management functions 11
    ms_addtoconf(_) 23
    ms_delconf(_) 31
    ms_estconf(_) 38
    ms_getcde(_) 64
    ms_getcnflist(_) 67
    ms_monconf(_) 81
    ms_remfromconf(_) 86
    ms_setcde(_) 106
    ms_unmonconf(_) 128
conference properties structure 139
configuration functions 12
    ms_getbrdparm(_) 62
    ms_getevt(_) 71
    ms_getevtmsk(_) 74
    ms_ResultMsg( ) 89
    ms_ResultValue( ) 92
    ms_setbrdparm(_) 99
    ms_setevtmsk(_) 109
    ms_setstparm(_) 115
connection extender 42
connection identifier 42
CT_DEVINFO data structure 134

D

data structures
    CT_DEVINFO 134
    MS_CADENCE 137
    MS_CDT 139
    MS_DataInfo 141
    MS_NCB 143
    SC_TSINFO 144
device management functions 12
    ms_close(_) 29
    ms_open(_) 84
    ms_stopfn(_) 119
diagnostic functions 13
    ms_tstcom(_) 121
    ms_tstdat(_) 123
Dialogic Integrated boards 15
distinctive ring 54
DM3 boards 15

E

events 131
extended attribute functions 15
    ATMS_STATINFO(_) 18
    ATMS_TSSGBIT(_) 20
extended connection functions 13
    ms_chgxtder(_) 26
    ms_delxtdcon(_) 33
    ms_estxtdcon(_) 42

H

HDSI boards 15

M

ms_addtoconf(_) 23
MS_CADENCE data structure 137
MS_CDT data structure 139
ms_chgxtder(_) 26
ms_close 29
MS_DataInfo data structure 141
ms_delconf(_) 31
ms_delxtdcon(_) 33
ms_dsprescount(_) 35
ms_estconf(_) 38
ms_estxtdcon(_) 42
ms_genring(_) 45
ms_genringCallerID( ) 50
ms_genringex(_) 54
ms_genziptone 60
ms_getbrdparm(_) 62

ms_getcde(_) 64
ms_getcnflist(_) 67
ms_getctinfo(_) 69
ms_getevt(_) 71
ms_getevtmsk(_) 74
ms_getxmitslot( ) 76
ms_listen( ) 78
ms_monconf(_) 81
MS_NCB data structure 143
ms_open(_) 84
ms_remfromconf(_) 86
ms_ResultMsg( ) 89
ms_ResultValue( ) 92
ms_SendData( ) 95
ms_setbrdparm(_) 99
ms_setcde(_) 106
ms_setevtmsk(_) 109
ms_SetMsgWaitInd( ) 113
ms_setstparm(_) 115
ms_setvol(_) 117
ms_stopfn(_) 119
ms_tstcom(_) 121
ms_tstdat(_) 123
ms_unlisten( ) 126
ms_unmonconf(_) 128

N

notification tone characteristics structure 143

P

platform
    support 15

R

ring cadence, distinctive 54

S

SC_TSINFO data structure 144
Springware boards 15
station functions 14
    ms_genring(_) 45
    ms_genringCallerID( ) 50
    ms_genringex(_) 54
    ms_genziptone(_) 60
    ms_SendData( ) 95
    ms_SetMsgWaitInd( ) 113
    ms_setvol(_) 117
supported boards 15
supported platform 15

T

TDM bus routing functions 13
    ms_getxmitslot( ) 76
    ms_listen( ) 78
    ms_unlisten( ) 126
TDM bus time slot information structure 144