MSI/SC Software Reference

for Windows NT

Copyright © 1997 Dialogic Corporation

PRINTED ON RECYCLED PAPER

COPYRIGHT NOTICE

Copyright 1997 Dialogic Corporation. All Rights Reserved.

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation. Every effort is made to ensure the accuracy of this information. However, due to ongoing product improvements and revisions, Dialogic Corporation cannot guarantee the accuracy of this material, nor can it accept responsibility for errors or omissions. No warranties of any nature are extended by the information contained in these copyrighted materials. Use or implementation of any one of the concepts, applications, or ideas described on Web pages maintained by Dialogic-may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not condone or encourage such infringement. Dialogic makes no warranty with respect to such infringement, nor does Dialogic waive any of its own intellectual property rights which may cover systems implementing one or more of the ideas contained herein. Procurement of appropriate intellectual property rights and licenses is solely the responsibility of the system implementor. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

All names, products, and services mentioned herein are the trademarks or registered trademarks of their respective organizations and are the sole property of their respective owners. DIALOGIC (including the Dialogic logo), DTI/124, SpringBoard, and Signal Computing System Architecture (SCSA) are registered trademarks of Dialogic Corporation. The following are also trademarks of Dialogic Corporation Trademarks can be found at http://www.dialogic.com/legal.htm (copyright link in lower right corner of any Dialogic website page) Board Locator Technology, D/41ESC, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, D/320SC, DIALOG/HD, GammaFax CP-4/SC, MSI/SC, SCbus, SCSA, Signal Computing System Architecture, SpringWare, Voice Driver, VFX/40ESC, and World Card.

IBM is a registered trademark, and IBM PC is a trademark of International Business Machines Corporation.
Windows NT is a registered trademark of Microsoft Corporation.

Publication Date: July, 1997

Part Number: 05-0505-002

Dialogic Corporation
1515 Route 10
Parsippany NJ 07054

Technical Support
Phone: 973-993-1443
Fax: 973-993-8387
BBS: 973-993-0864
Email: CustEng@dialogic.com

For **Sales Offices** and other contact information, visit our website at **http://www.dialogic.com**

Table Of Contents

List Of Tables

List of Figures

About This MSI/SC Guide

Products Covered by this Guide

The MSI/SC refers to the Dialogic modular station interface boards designed for SCbus support only. This guide covers the software for the products listed in the table below.

Model	Description
MSI/80SC	Baseboard-only product with 8 station interfaces.
MSI/160SC	Baseboard product with 1 daughterboard module (16 station interfaces).
MSI/240SC	Baseboard product with 2 daughterboard modules (24 station interfaces).
MSI/SC-R	The MSI/80SC, MSI/160SC, or the MSI/240SC with ringing capability.

NOTE: All boards listed in the table above have conference capability.

Product Terminology

The following product naming conventions are used throughout this guide:

D/41ESC refers to the Dialogic 4-channel voice board with on-board analog loop start interface.

D/160SC-LS refers to the Dialogic 16-channel voice board with on-board analog loop start interface.

D/240SC refers to the Dialogic 24-channel voice board for use with a network interface board.

D/240SC-T1 refers to the Dialogic 24-channel voice board with on-board T-1 digital interface.

D/300SC-E1 refers to the Dialogic 30-channel voice board with on-board E-1 digital interface.

D/320SC refers to the Dialogic 32-channel voice board for use with a network interface board.

D/xxxSC refers to voice and telephone network interface resource boards that communicate via the SCbus. These boards include D/41ESC, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, and D/320SC.

DIALOG/HD or **SpanCard** refers to voice and telephone network interface resource boards that communicate via the SCbus. These boards include D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, and D/320SC.

MSI refers to the Dialogic modular station interface board for the PCM Expansion Bus (PEB).

MSI/SC refers to the Dialogic modular station interface product for the SCbus.

MSI/SC-R refers to the Dialogic station interface product with ringing capability.

SCbus is the TDM (Time Division Multiplexed) bus connecting SCSA (Signal Computing System Architecture) voice, telephone network interface, and other technology resource boards together.

SpanCard - same as **DIALOG/HD**.

VFX/40ESC is a Dialogic SCbus voice and fax resource board with on-board analog loop-start interfaces. The VFX/40ESC board consists of a D/41ESC baseboard and a FAX/40E daughterboard that provides four channels of enhanced voice and fax services in a single slot. Throughout this document, all references to the D/41ESC board apply to the D/41ESC baseboard component of the VFX/40ESC board.

For additional information on these products, refer to the manuals listed in *Appendix B*.

How to Use this Guide

This guide is written for users who have purchased a Dialogic MSI/SC board and related software for installation on a PC operating in a Windows NT environment.

The following steps explain the order in which an MSI/SC board and Dialogic software for Windows NT should be installed, checked, and programmed:

1. Prepare the MSI/SC board for installation using the appropriate hardware quick installation card (see *Appendix B*).
2. Install the necessary device drivers following the procedure described in the *System Release Software Installation Reference for Windows NT*.
3. Install the MSI/SC board in your PC following the procedure in the hardware quick installation card (see *Appendix B*).
4. Refer to this MSI/SC *Software Reference for Windows NT* to develop application programs.

To use software for other Dialogic devices, refer to the appropriate software reference for specific instructions (see *Appendix B*).

Organization of this Guide

This guide is organized as follows:

Chapter 1 provides a description of the MSI/SC board and presents an overview of the Dialogic modular station interface technology.

Chapter 2 provides an overview of the MSI/SC device driver library functions.

Chapter 3 is an alphabetical reference to the MSI/SC library functions. It includes a detailed description and a programming example for each function.

Chapter 4 provides brief guidelines for developing applications.

Appendix A lists entries and returns for the Dialogic Standard Runtime Library (SRL). For more information, refer to the *Standard Runtime Library Programmer's Guide* in the *Voice Software Reference for Windows NT.*

Appendix B lists related publications for further information on the MSI/SC product and other Dialogic products.

A Glossary and an Index follow the appendices.

1. MSI/SC Introduction

MSI/SC Product Overview

The Dialogic MSI/SC board is a modular station interface adding analog devices, such as modems, fax machines, and audio equipment, into new or existing systems based on the Dialogic SCbus architecture. With the MSI/SC board, users can extend the range of their inbound and outbound telemarketing centers.

The MSI/SC product series consists of a baseboard that includes 8 integrated station interfaces and allows up to 2 additional daughterboard modules, each with an additional 8 station interfaces. The baseboard-only product is referred to as an MSI/80SC. A baseboard with 1 daughterboard module is referred to as an MSI/160SC (16 station interfaces), and a baseboard with 2 daughterboard modules is referred to as an MSI/240SC (24 station interfaces).

The MSI/SC feature set is based on Dialogic's MSI-C board; however, its hardware architecture is based on the D/41E board. This allows the MSI/SC to take advantage of the D/41E BLT circuit, programmable interrupts, and shared RAM interface. In addition, the MSI/SC board provides a ring option, making it capable of generating AC voltage sufficient to ring standard 2500 type telephones.

1.1. Typical Applications

The MSI/SC board and software allows an application program operating in the host PC to communicate between SCbus-compatible devices and analog station devices. Applications for the MSI/SC board include:

- Inbound/outbound telemarketing
- Operator services such as billing automation, directory assistance, and intercept treatments
- Customer service
- Automatic call distribution (ACD)
- Dictation/transcription
- Local information services

Conferencing resources serve the SCbus with advanced features such as:

- Two to eight-party conferencing
- Up to 32 resources of total conferencing (4 to 16 conferences)
- Conferences of any combination of stations and network channels
- Hidden training for smooth entry of new conferees without disruptive training noise
- Monitoring an agent without disrupting the conversation
- Coaching feature to allow a supervisor to speak to an agent without the client hearing the supervisor. The client can hear the agent at all times (no switching)
- Tone generation:

  Zip tone indicates incoming call to agents using headsets

  Notification tones when a party is added to or removed from a conference (as required by the law in many states)
- User programmable periodic notification tones to indicate units of time that expired during a call
- Programmable volume control for station devices

Refer to *Section 1.3. Conferencing* for more information

1.2. Compatibility

The MSI/SC board feature set is based on the Dialogic PEB-based MSI-C board. The MSI/SC hardware is based on the Dialogic D/41E board. This hardware design enables the MSI/SC to take advantage of the Board Locator Technology (BLT) circuit, programmable interrupts, and shared RAM interface.

NOTE: The MSI/SC board is an SCbus-only product, and does not support PEB.

Using the MSI/SC, developers can build large call center configurations without the need for DMX boards or crossover cables. Other significant differences between the MSI/SC and the MSI-C board include:

- use of the 1024 time slots available on the SCbus
- ability to ring 2500 type telephones
- on-board ring voltage generator
- relay interlock for loop and ring voltages

Conference-specific differences between the MSI/SC board and the MSI-C board include:

- pupil/coach feature
- hidden training
- extended connections

1.3. Conferencing

The MSI/SC board features up to 32 resources of total conferencing with up to eight parties in a conference.

MSI/SC conferencing provides the following features to your application:

Monitor mode	This feature allows a conference to be monitored by several people without interrupting the conference.
Coach	Typically, a supervisor. The coach, while monitoring a conversation between a pupil and a client, can talk to the pupil in confidence. The pupil, however, cannot reply in confidence to the coach.
Pupil	Typically, the agent who is heard by all parties in the conference. The pupil is the only conference participant who hears the coach.

NOTE: A conference may include only one coach and one pupil at any given time. There may be more than one client, but conference size is limited to the maximum number of participants permitted in the conference.

1.4. Extended Connections

An extended connection allows a coach to join a connection at anytime without interrupting the conversation between the agent and the client.

A connection is defined as a full-duplex, SCbus routing between two parties. One party must be a station on the MSI/SC board.

An extended connection is a connection where there is a third party. This third party, herein referred to as the *connection extender*, can always hear what the other two parties are saying. The connection extender's input in the connection is application defined. If the connection extender has no input, it is in monitor mode. If the connection extender can talk to only one party, designated as the pupil, it is in coach mode. If the connection extender can talk to both members of the connection, it is in participant mode.

NOTES:

1. A connection may be set up using the SCbus routing convenience function **nr_scroute( )** (see the *SCbus Routing Function Reference Guide for Windows NT*).
2. It is the application's responsibility to set up a connection prior to extending a connection. The MSI/SC software does not check for the presence of a connection between parties in order to extend it.

1.5. Resource Allocation

The DSP on the MSI/SC board has 32 resources managed by the application. Calling any of the following functions will cause the available resource count to change:

Function	Condition
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_estconf( )	Uses the total number of parties in the conference.
ms_addtoconf( )	Uses one resource every time a party is added to a conference.

Function	Condition
ms_remfromconf( )	Frees one resource.
ms_delconf( )	Frees all resources in use by the conference.
ms_monconf( )	Uses one resource.
ms_unmonconf( )	Frees one resource.
ms_estxtdcon( )	Uses three resources.
ms_delxtdcon( )	Frees three resources.

NOTES: **1.** The channel selector of the party does not affect the resource usage.

2. A conference is limited to eight parties. A monitor is counted as one of the eight parties.

3. When zip tone support is enabled, 31 conferencing resources will be available.

1.6. Functional Description

The MSI/SC baseboard and each daughterboard contain eight line interfaces and eight COder/DECoders (CODECs). Each line interface provides loop-start current to one 2500 series equivalent station device.

The line interface also separates the inbound signal into an audio signal which is sent to the CODEC, and an on-hook/off-hook signal which is forwarded by the control processor to the application program.

The CODEC converts inbound audio to 8-bit PCM data and outbound PCM data to analog audio. The CODEC gain may be individually set for each station device.

A cross-point switch on the MSI/SC board routes PCM data between the station devices, the DSP and the SCbus.

The MSI/SC board conferencing feature allows conferences to be established between SCbus time slots and/or station interfaces.

The control microprocessor executes commands received from the host PC, and controls all operations of the MSI/SC board. Communications between the control microprocessor and the host PC is accomplished via a shared RAM interface mechanism.

Those operations demanding real time response are interrupt driven. All MSI/SC boards installed in the PC share the same interrupt line. When the system is initialized, firmware to control all board operations is downloaded from the host PC to the on-board RAM. The downloadable firmware enables easy feature enhancements in the future.

The Board Locator Technology (BLT) circuit operates in conjunction with a rotary switch to determine and set non-conflicting slot and IRQ interrupt-level parameters. This feature eliminates the need to set jumpers or DIP switches.

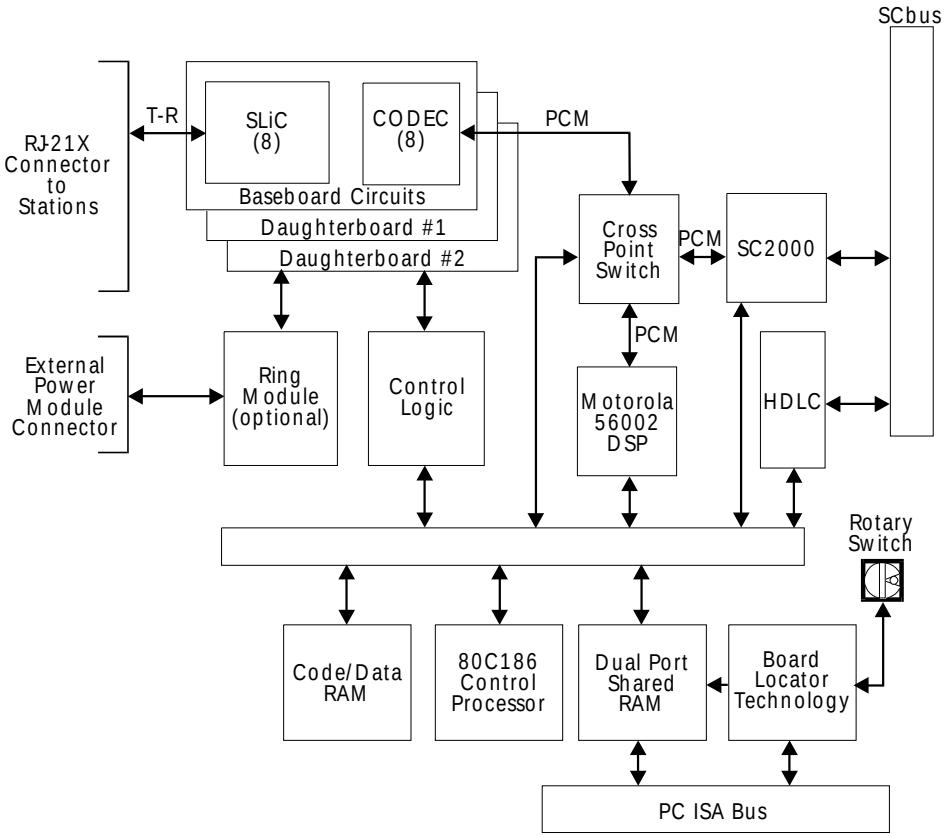

Figure 1. The MSI/SC Board

2. MSI/SC Library Function Overview

2.1. Library Function Categories

The Dialogic MSI/SC Windows NT library provides support for the Dialogic MSI/SC boards. This chapter provides an overview of the library functions. A detailed reference for these functions is located in Chapter 3.

The MSI/SC library functions provide the necessary building blocks to create MSI/SC applications. These functions can be divided into the following categories:

Attribute	• retrieve device-specific information
Conference Management	• control the conferencing features
Configuration	• set and retrieve the MSI/SC device parameters
Device Management	• open and close devices
Diagnostic	• test the MSI/SC device
Extended Connection	• control extended connections
Station	• control station interfaces

NOTE: Some MSI/SC Windows NT library functions can operate in either synchronous or asynchronous mode. 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. Refer to *Appendix A* for more information.

The MSI/SC SCbus Routing Functions can be found in the *SCbus Routing Function Reference for Windows NT*. Refer to the *SCbus Routing Guide* for an explanation of SCbus routing.

NOTE: In order to use the SCbus routing convenience functions **nr_scroute( )** and **nr_unscroute( )** with the MSI/SC device, the preprocessor directive DTISC must be defined using the preprocessor option IDs when compiling the *sctools.c* file.

Each category and its functions are briefly described in the following sections.

2.1.1. Attribute Functions

ms_dsprescount( )	• returns DSP resource count
ms_getctinfo( )	• returns information about the station interface device

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

2.1.2. Conference Management Functions

ms_addtoconf( )	• adds a party to an existing conference
ms_delconf( )	• deletes a conference
ms_estconf( )	• establishes a conference
ms_getcde( )	• gets conference descriptor table
ms_getcnflist( )	• gets conferee list
ms_monconf( )	• adds a monitor to a conference
ms_remfromconf( )	• removes a party from a conference
ms_setcde( )	• changes conference descriptor table
ms_unmonconf( )	• removes a monitor from a conference

These functions are used to manage all conference activities.

2.1.3. Configuration Functions

ms_getbrdparm( )	• returns board parameters
ms_getevt( )	• retrieves an unsolicited event
ms_getevtmsk( )	• returns the station event mask
ms_setbrdparm( )	• changes board parameters
ms_setevtmsk( )	• changes station event mask

ms_setstparm( )	• changes station level parameters

These functions set the MSI/SC device parameters and event masks and check the status of the MSI/SC device parameter settings.

2.1.4. Device Management Functions

ms_close( )	• closes MSI/SC device
ms_open( )	• opens MSI/SC device
ms_stopfn( )	• stops a multitasking function in progress

ms_open( ) and **ms_close( )** open and close devices, respectively. **ms_stopfn( )** is invoked to stop a multitasking function in progress.

2.1.5. Diagnostic Functions

ms_tstcom( )	• runs communications test
ms_tstdat( )	• runs data test

Diagnostic functions check the functionality of the MSI/SC firmware and hardware. The **ms_tstcom( )** function tests if the PC can communicate with the MSI/SC board. The **ms_tstdat( )** function tests if data is passed successfully between the PC and the MSI/SC board.

2.1.6. Extended Connection Functions

ms_chgxtder( )	• changes the attributes of the connection extender
ms_delxtdcon( )	• deletes the extended connection
ms_estxtdcon( )	• establishes an extended connection

These functions are used to manage all extended connection activities.

2.1.7. Station Functions

ms_genring( )	• generates a ring to a station
ms_genziptone( )	• generates zip tone to a station
ms_setvol( )	• sets station volume

ms_setvol( )controls the station interface volume and **ms_genring( )** generates ringing to the station. **ms_genziptone( )** generates zip tone to a station.

NOTE: **ms_genring( )** is only supported on the MSI/SC-R boards.

2.2. Extended Attribute Functions

ATMS_DNLDVER( )	• returns the downloaded firmware version
ATMS_STATINFO( )	• gets the station information on the MSI/SC
ATMS_TSSGBIT( )	• gets channel signaling bit status

Attribute functions return information about the specified device. **Standard Attribute** functions, which are contained in the Dialogic Standard Runtime Library (see *Appendix A*), provide generic information about a device, such as its name, or the last error that occurred on the device. **Extended Attribute** functions return information that is specific to the device.

Errors for Extended Attribute functions are handled in the same way as all other functions described in this chapter. Refer to *Section 2.3. Error Handling* for information about retrieving the errors.

2.3. Error Handling

All the MSI/SC library functions return a value that indicates the success or failure of the function call. MSI/SC library functions can return one of the following values:

0	function success

AT_FAILURE function error

If a function fails, the error can be retrieved using the Standard Runtime Library (SRL) **ATDV_LASTERR( )** function.

NOTES:

1. The function **ms_open( )** is the exception to the above error-handling rules. A **ms_open( )** function call returns a device handle if the function call is successful. A device handle is a non-zero value. If **ms_open( )** fails, the return code is AT_FAILURE and the specific error is found in the global variable **errno** defined in *errno.h*.
2. The Standard Attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** can be used to obtain the last error that occurred on a device. Refer to *Appendix A* for more information.
3. If the error returned by **ATDV_LASTERR( )** is **E_MSSYSTEM**, a Windows NT system error has occurred. Check the global variable **errno** defined in *errno.h*.

Some of the causes and values of error codes that may be returned to the application by the MSI/SC board are identical to those used for Dialogic Digital Network Interface products. The following tables list possible error codes for all Network boards and MSI/SC-specific error codes.

Table 1. Error Types Defined in dtilib.h

Error Returned	Description
EDT_SH_LIBBSY	Switching Handler Library is busy.
EDT_SH_BADINDX	Invalid Switching Handler index number.
EDT_SH_LIBNOTINIT	Switching Handler Library has not been initialized.
EDT_SH_NOCLK	Switching Handler Clock fallback failed.
EDT_SH_MISSING	Switching Handler is not present.
EDT_SH_BADLCLTS	Invalid local time slot number.
EDT_SH_BADTYPE	Invalid local time slot type.

Table 2. Error Types Defined in msilib.h

Error Returned	Description
E_MSABORT	Abort received response.
E_MSADDRS	Incorrect address.
E_MSBADBRDERR	Board is missing or defective.
E_MSBADCMDERR	Invalid or undefined command to driver.
E_MSBADCNT	Incorrect count of bytes requested.
E_MSBADGLOB	Incorrect global parameter number.
E_MSBADPORT	First byte appeared on reserved port.
E_MSBADVAL	Invalid parameter value passed in value pointer.
E_MSCHKSUM	Incorrect checksum.
E_MSDATTO	Data reception timed out.
E_MSDTTSTMOD	In test mode; cannot set board mode.
E_MSFWERR	Firmware returned an error.
E_MSINVBD	Invalid board.
E_MSINVMSG	Invalid message.
E_MSINVTS	Invalid time slot.
E_MSMBFMT	Wrong number of bytes for multiple byte request.
E_MSMBIMM	Received an immediate termination.
E_MSMBINV	First byte appeared on data port.
E_MSMBOVR	Message was too long, overflow.
E_MSMBPORT	Received multiple byte data on port other than 0 or 1.
E_MSMBTERM	Terminating byte other than FEH or FFH.
E_MSMBUND	Under the number of bytes for a multibyte request.
E_MSMSGCNT	Count received did not match actual count.
E_MSNOCLK	No clock source present.
E_MSNOIDLEERR	Time slot is not in idle/closed state.
E_MSNOMEMERR	Cannot map or allocate memory in driver.
E_MSNOTDNLD	Not downloaded.
E_MSPARAMERR	Invalid parameter.
E_MSRANGEERR	Bad/overlapping physical memory range.
E_MSSIGINS	Insertion signaling not enabled.
E_MSSIGTO	Transmit/receive did not update in time.
E_MSSIZEERR	Message too big or too small.
E_MSSKIPRPLYERR	A required reply was skipped.
E_MSSTARTED	Cannot start when already started.
E_MSSUCC	No error.
E_MSSYSTEM	Windows NT system error - check the global variable

Error Returned	Description
	errno for more information.
E_MSTMOERR	Timed out waiting for reply from firmware.
E_MSTSASN	Time slot already assigned.
E_MS1PTY	Cannot remove party from one party conference.
E_MSBADCHPARM	Invalid channel parameter number.
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_MSGLOBREAD	Cannot read parameter globally.
E_MSINVCB	Invalid control block ID.
E_MSINVCATTR	Invalid conference attribute selector.
E_MSINVCNF	Invalid conference number.
E_MSINVDSP	Invalid DSP specified.
E_MSINVMT	Invalid multitasking function.
E_MSINVPATTR	Invalid party attribute.
E_MSINVPTNUM	Invalid party number.
E_MSINVPTYCNT	Invalid number of parties specified.
E_MSINVPTYTYPE	Invalid conference member type.
E_MSINVRNGCNT	Invalid number of ring counts.
E_MSINVST	Invalid station.
E_MSINVVAL	Bad global parameter value.
E_MSINVTS	Invalid time slot 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_MSNOCT	Station not connected.
E_MSNODSPTS	All time slots going to the DSP are busy.
E_MSNOFEMCH	No MSI/SC daughterboard to support this channel.
E_MSNOMON	No monitor exists for this conference.
E_MSNONCNFCH	Channel not assigned to specified conference.
E_MSNOTS	No time slot assigned to channel.
E_MSNOTSALLOC	No time slots allocated to the board.
E_MSPTYASN	Party already assigned.
E_MSSNDZIP	Sending a zip tone to this station.

Error Returned	Description
E_MSSTASN	Time slot already assigned to station.
E_MSSYSTEM	System error- see **erro** 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_MSZIPON	Station is currently "zipping."
E_MSZIPEN	Zip tones disabled - message not allowed.

2.4. Include Files

Function prototypes and equates are defined in *dtilib.h* and *msilib.h*. Applications that use the MSI/SC Windows NT library functions for MSI/SC support must include the following statements:

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

To perform error handling in your routines, your source code must include the following line:

```
#include <errno.h>
```

Code that uses Voice boards with the current version of the Windows NT Voice Driver must include the following statements in the order shown:

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

The current version of the Windows NT voice driver requires that *srllib.h* precedes any other Dialogic header file include statements.

3. MSI/SC Function Reference

This chapter contains an alphabetical listing of the Dialogic MSI/SC library functions used to interface with the MSI/SC board. For information about Standard Attribute functions, refer to *Appendix A*. For information on the SCbus Routing functions, refer to *the SCbus Routing Function Reference for Windows NT*.

3.1. Documentation Conventions

Each function is listed in alphabetical order and provides the following information:

Function Header	Located in the beginning of each function. The header lists the function name, function syntax, input parameters, outputs or returns, includes, category, and mode (asynchronous and/or synchronous). The function syntax and inputs are shown using standard C language syntax.
Description	Provides a detailed description of the function operation, including parameter descriptions.
Cautions	Provides warnings and reminders.
Example	Provides one or more C language coding examples showing how the function can be used.
Errors	Lists specific error codes for each function.
See Also	Provides a list of related functions.

Name:	long ATMS_DNLDVER(devh)
Inputs:	int devh • MSI/SC board device handle
Returns:	version of MSI/SC firmware used by the device AT_FAILURE on failure
Includes:	srllib.h dtilib.h msilib.h
Category:	Extended Attribute
Mode:	synchronous

■ Description

The **ATMS_DNLDVER( )** function returns the MSI/SC firmware version that was downloaded to the device specified in **devh**. This number is returned in the standard Dialogic version numbering format.

Parameter	Description
devh:	The MSI/SC board device handle returned by a call to **ms_open( )**.

■ Dialogic Version Numbering

A Dialogic version number consists of two parts that provide:

- The release TYPE
  (For example: Production or Beta)
- The release NUMBER, consisting of different elements depending on the type of release.

  Example: 1.00 Production
  1.00 Beta 5

NOTE: The examples above are shown in the convention used by Dialogic to display version numbers.

This function returns the version number as a long integer (32 bits) in binary coded decimal format. *Table 3* shows the values returned by each nibble in the long integer.

Table 3. Returns for Release Type

Nibble (4 bits)				
1	**2**	**3 & 4**	**5 & 6**	**7 & 8**
TYPE	**PRODUCTION RELEASE NUMBER**		**INTERNAL NUMBER**	
Production	Major Release No.	Minor Release No.	N/A	N/A
Beta	Major Release No.	Minor Release No.	Beta Number	N/A

Major and Minor Release Numbers

Major and minor release numbers distinguish major revisions from minor revisions to production releases. The major number converts to a single digit integer that increases with each major revision to the release. The minor number converts to a two digit integer that increases with each minor revision to the release.

In decimal number format, the major number is the number before the decimal point, and the minor number is the number after the decimal point.

The following list gives examples of each type of release. The values used in these examples have been converted from the binary coded decimal numbers returned in the long integer and are displayed according to Dialogic convention.

1.00 Production
1.00 Beta 5

■ Cautions

The function fails if an invalid device handle is passed.

■ Example

```
#include <windows.h>
#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 */
   long version;   /* Version number of firmware */

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

   /*
    * Get the version number of the firmware
    */
   version = ATMS_DNLDVER( bddev );
   if ( version == AT_FAILURE ) {
      do_error( bddev, "ATMS_DNLDVER()" );
      exit( 1 );
   }
   /*
    * Display it
    */
   printf( "MSI/SC Download version number is %d.%02x\n",
           (int)((version>>24L)&0x0F), ((version >>16L)&0xFF) );

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

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

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

Name: long ATMS_STATINFO (devh,statinfop)

Inputs:
- int devh — MSI/SC board device handle
- char * statinfop — pointer to four bytes containing station information

Returns: station information
AT_FAILURE on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Extended Attribute

Mode: synchronous

■ Description

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

Parameter	Description
devh:	The valid MSI/SC board device handle returned by a call to **ms_open( )**.
statinfop:	Pointer to four bytes. When the function returns, the first byte will contain the total number of stations on the MSI/SC board. Bytes two through four will indicate the status of the baseboard and two daughterboards, respectively.

■ Cautions

This function fails if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#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("msiB1C1",0)) == -1) {
    printf( "Cannot open MSI B1, errno=%d", errno);
    exit(1);
}

/*
 * Continue processing
 */
/* 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=1;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);
}
```

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

Name: long ATMS_TSSGBIT(devh)
Inputs: int devh • MSI/SC station device handle
Returns: state of channel
AT_FAILURE on failure
Includes: srllib.h
dtilib.h
msilib.h
Category: Extended Attribute
Mode: synchronous

■ Description

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

Parameter	Description
devh:	The MSI/SC station device handle returned by a call to **ms_open( )**.

The returned bitmask represents the following:

MS_ONHOOK	MSI/SC station is on-hook
MS_OFFHOOK	MSI/SC station is off-hook

These equates are defined in *msilib.h*.

■ Cautions

This function fails if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      printf( "Cannot open station msiB1C1. errno = %d", errno );
      exit( 1 );
   }

   /*
    * Get station signaling bits
    */
   tsbits = ATMS_TSSGBIT( tsdev );
   if ( tsbits == AT_FAILURE ) {
      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 ) == AT_FAILURE ) {
      printf( "Cannot close station msiB1C1. errno = %d", errno );
   }
}
```

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

Name:	int ms_addtoconf(devh,confID,cdt)	
Inputs:	int devh	• MSI/SC board device handle
	int confID	• conference identifier
	MS_CDT *cdt	• pointer to conference descriptor table
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ 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.

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier number.
cdt:	Pointer to the conference descriptor table.

NOTES:

1. 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 a problem since the coach usually will not need to speak before some conversation has taken place between the client and the pupil.
2. Only one party at a time can be added using this function.
3. Successfully invoking this function causes a conferencing resource to be used when a party is successfully added to a conference.

The MS_CDT structure has the following format:

```
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;
```

The chan_num denotes the station number or SCbus time slot number of the device to be included in the conference. The chan_sel defines the meaning of chan_num. Valid choices are as follows:

- MSPN_STATION MSI/SC station number
- MSPN_TS SCbus time slot number

Channel attribute is a bitmask describing the party's properties within the conference. Valid choices are:

- MSPA_NULL No special attributes for party.
- MSPA_RO Party participates in conference in receive-only mode.
- MSPA_TARIFF Party receives periodic tone for duration of call.
- MSPA_COACH Party is a coach. Coach heard by pupil only.
- MSPA_PUPIL Party is a pupil. Pupil hears everyone including coach.

Table 4. Valid Attribute Combinations

Pupil	Coach	Periodic Tone	Receive-only mode
			X
		X	
		X	X
	X		
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. For SCbus time slot members of a conference, the number of the time slot to listen to (MSPN_TS) is returned in the chan_lts field. This information is used by the application to listen to the conferenced signal. This is not applicable to MSI/SC stations because the stations (MSPN_STATION) do not use SCbus time slots.

The chan_attr field in the CDT structure is redefined as follows:

```
#define   chan_lts     chan_attr
```

NOTE: The **cdt** structure is reused to return the listen SCbus 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.

Example

```
#include <windows.h>
#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_tsarray = &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_tsarray = &cdt[0].chan_lts;

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

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name:	int ms_chgxtder(devh,xid, cdt)	
Inputs:	int devh	• MSI/SC board device handle
	int xid	• extended connection identifier
	MS_CDT *cdt	• pointer to descriptor table
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Extended Connection	
Mode:	synchronous	

■ 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.

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

Parameter	Description
devh:	The MSI/SC board device handle.
xid:	The extended connection identifier.
cdt:	Pointer to the conference descriptor table element.

The characteristics of the connection extender are described by the MS_CDT structure:

```
typedef struct {
    int    chan_num;
    int    chan_sel;
    int    chan_attr;
} MS_CDT;
```

The chan_num denotes the station number or the SCbus time slot number of the connection extender. The chan_sel can be one of the following:

- MSPN_STATION MSI/SC station number

- MSPN_TS SCbus time slot number

Channel attribute is a bitmask describing the connection extender's properties within the extended connection. The valid values are:

- MSPA_TARIFF Party receives a periodic tone for the duration of call.
- MSPA_NULL No special attributes for party.
- MSPA_RO Party participates in conference in receive-only mode.
- MSPA_COACH Party is a coach. Coach heard by pupil only.
- MSPA_PUPIL Party is a pupil. Pupil hears everyone including coach.

Table 5. Valid Attribute Combinations

Pupil	Coach	Periodic Tone	Receive-only mode
			X
		X	
		X	X
	X		
X			
X		X	

NOTES: **1.** Only one coach and one pupil are allowed in an extended connection.

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

3. The signal that the connection extender should listen to is always present on the SCbus, irrespective of the chan_sel of the connection extender.

■ Cautions

This function fails when:

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

■ Example

```
#include <windows.h>
#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_tsarray = &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);

}
```

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name: int ms_close(devh)
Inputs: int devh • MSI/SC device handle
Returns: 0 on success
AT_FAILURE on failure
Includes: srllib.h
dtilib.h
msilib.h
Category: Device Management
Mode: synchronous

■ Description

The **ms_close( )** function closes the MSI/SC device previously opened by the calling process and **ms_open( )**. The devices are either MSI/SC 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/SC device handle returned by a call to **ms_open( )**.

■ Cautions

1. This function fails if the device handle is invalid.
2. The **ms_close( )** function affects only the link between the calling process and the device. Other processes are unaffected by **ms_close( )**.
3. If event notification is active for the device to be closed, call the SRL **sr_dishdlr( )** function prior to calling **ms_close( )**.
4. A call to **ms_close( )** does not affect the configuration of the MSI/SC.
5. Dialogic devices should never be closed using the Windows NT **close( )**.

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      printf( "Cannot open board msiB1. errno = %d\n", errno );
      exit( 1 );
   }

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

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

}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_open( )**

Name:	int ms_delconf(devh, confID)	
Inputs:	int devh	• MSI/SC board device handle
	int confID	• conference identifier
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ 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/SC board device handle.
confID:	The MSI/SC 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.

■ Example

```
#include <windows.h>
#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 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_addtoconf( )**
- **ms_estconf( )**
- **ms_monconf( )**

- **ms_remfromconf( )**
- **ms_unmonconf( )**

Name:	int ms_delxtdcon(devh,xid)	
Inputs:	int devh	• MSI/SC board device handle
	int xid	• extended connection identifier
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Extended Connection	
Mode:	synchronous	

■ 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 in conversation by virtue of SCbus routing will still remain in a connection.

Parameter	Description
devh:	The MSI/SC 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/SC board.
- The connection ID is invalid.

■ Example

```
#include <windows.h>
```

```
#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

 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_chgxtder( )**
- **ms_estxtdcon( )**

Name:	int ms_dsprescount(devh,valuep)	
Inputs:	int *devh	• MSI/SC board device handle
	int *valuep	• pointer to the memory location to receive the free DSP resource count
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Attribute	
Mode:	synchronous	

■ Description

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

Parameter	Description
devh:	The MSI/SC board device handle.
valuep:	Pointer to the location to contain the free DSP resource count.

Each DSP has 32 resources managed by the application. Calling any of the following functions may cause the available resource count to change.

Function	Condition
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_estconf( )	Uses the total number of parties in the conference.
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_monconf( )	Uses one resource.

ms_unmonconf( )	Frees one resource.
ms_estxtdcon( )	Uses three resources.
ms_delxtdcon( )	Frees three resources.

NOTES: **1.** A conference is limited to eight parties. A monitor is counted as one of the eight parties.

2. When zip tone support is enabled, 31 conferencing resources will be available.

■ Cautions

This function fails when the device handle specified is invalid.

■ Example

```
#include <windows.h>
#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);
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name:	int ms_estconf(devh,cdt,numpty,confattr,confID)	
Inputs:	int devh	• MSI/SC board device handle
	MS_CDT *cdt	• pointer to conference descriptor table
	int numpty	• number of parties in conference
	int confattr	• conference attributes
	int *confID	• pointer to memory location to receive the conference identifier
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ Description

The **ms_estconf( )** function establishes a conference of up to four parties.

Parameter	Description
devh:	The MSI/SC board device handle.
cdt:	The pointer to the conference descriptor table.
numpty:	Number of parties in the conference.
confattr:	The conference attributes.
confID:	Pointer to the memory location containing the conference ID number.

NOTES: **1.** 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 a problem since the coach usually will not need to speak before some conversation has taken place between the client and the pupil.

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

The conference descriptor table is an array of MS_CDT structures. The MS_CDT structure has the following format:

```
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;
```

The chan_num denotes the station number or SCbus time slot number of the device to be included in the conference. The chan_sel defines the meaning of the chan_num. Valid choices are as follows:

- MSPN_STATION MSI station number
- MSPN_TS SCbus time slot number

The chan_attr is a bitmask describing the party's properties within the conference. Valid choices are:

- MSPA_NULL No special attributes for party.
- MSPA_RO Party participates in conference in receive-only mode.
- MSPA_TARIFF Party receives periodic tone for duration of call.
- MSPA_COACH Party is a coach. Coach is heard by pupil only.
- MSPA_PUPIL Party is a pupil. Pupil hears everyone including coach.

Table 6. Valid Attribute Combinations

Pupil	Coach	Periodic Tone	Receive-only mode
			X
		X	
		X	X
	X		
X			
X		X	

NOTES: **1.** Only one coach and one pupil are allowed in a conference at any time. Specifying more than one of either will cause unexpected results.

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

Conference attribute is a 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* conferees 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.

For SCbus time slot members of a conference, the number of the time slot to listen to is returned in the chan_lts field.

The chan_attr field in the CDT structure is redefined as follows:

```
#define    chan_lts        chan_attr
```

This information is used by the application to listen to the conferenced signal. This is not applicable to MSI/SC stations because the stations do not use SCbus time slots.

NOTES: **1.** MSI/SC stations (those with chan_sel set to MSPN_STATION) do not use SCbus time slots.

2. This function may be used to establish a conference of up to 4 parties. **ms_addtoconf( )** must be used to increase the size of the conference beyond 4 and up to 8 parties.

3. The **cdt** structure is reused to return the listen SCbus time slot information. The application is responsible for maintaining the integrity of the data in the structure.

■ Cautions

Dialogic does not support any form of cascading conferences. If you attempt cascading conferences, conference quality may deteriorate significantly.

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 an SCbus time slot.

■ Example

```
#include <windows.h>
#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_tsarray = &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_tsarray = &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  */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name:	int ms_estxtdcon(devh,cdt,xid)	
Inputs:	int devh	• MSI/SC 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 AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Extended Connection	
Mode:	synchronous	

■ Description

The **ms_estxtdcon( )** function establishes an extended connection. An extended connection is a connection in which there is a third party.

Parameter	Description
devh:	The MSI/SC board device handle.
cdt:	Pointer to the conference descriptor table.
xid:	The pointer to the memory location containing the extended connection number.

For the purpose of this function, a connection is a full-duplex, SCbus routing between two parties. A connection may be set up using the convenience function **nr_scroute( )**.

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 an SCbus 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 an SCbus 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.

NOTES: **1.** Calling this function uses three resources.

2. 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 a problem since the coach usually will not need to speak before some conversation has taken place between the client and the pupil.

3. 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.

The extended connection is described by the descriptor table. A descriptor table is simply an array of MS-CDT structures. 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. The structure of each descriptor table entry is as follows:

```
typedef struct {
   int   chan_num;
   int   chan_sel;
   int   chan_attr;
} MS_CDT;
```

The chan_num denotes the station number or SCbus time slot number of a party within an extended connection. The chan_sel defines the meaning of chan_num. Valid choices are as follows:

- MSPN_STATION MSI/SC station number
- MSPN_TS SCbus time slot number

The chan_attr is a bitmask describing a party's properties within an extended connection. Valid choices for the attributes of the first entry (connection identifier) in the descriptor table are:

- MSPA_NULL No special attributes for party.
- MSPA_TARIFF Party receives periodic tone for duration of call.

- MSPA_PUPIL Party is a pupil. Pupil hears everyone including coach.

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.

Valid values for the second entry (connection extender) in the descriptor table are:

- MSPA_NULL Party can talk to members in extended connection.
- MSPA_RO Party participates in conference in receive only mode.
- MSPA_TARIFF Party receives periodic tone for duration of call.
- 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.

Table 7. Valid Attribute Combinations

AGC Disabled	Pupil	Coach	Periodic Tone	Receive-only mode
				X
			X	
			X	X
		X		
	X			
	X		X	
X	X			
X	X		X	

NOTES: **1.** Only one coach and one pupil are allowed in an extended connection.

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

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. We recommend that MSPA_NOAGC only be used when there is a pupil.

■ Cautions

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

■ Example

```
#include <windows.h>
#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_tsarray = &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);

}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

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
AT_FAILURE on failure

Includes: srllib.h
dxxxlib.h
dtilib.h
msilib.h

Category: Station

Mode: synchronous/asynchronous

■ 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.

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.

NOTES: **1.** Dialogic recommends specifying at least two rings. If you specify one ring, the phone may not ring.

2. A ring duty cycle includes an on time (ring generation) and off time (no ring). If **ms_genring( )** is received by the MSI/SC 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.
3. This function is only supported on MSI/SC-R boards.
4. **ms_genring( )** will fail when executed on a station currently off-hook. The error returned is E_MSBADRNGSTA.
5. A glare condition occurs when two parties seize the same line for different purposes. Although very rare, 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.

In asynchronous mode, 0 indicates that the function was initiated while AT_FAILURE indicates error. For successful completion of ringing, MSEV_RING will be returned. MSEV_NORING will be returned if the ring is not successful. The event data for MSEV_RING is as follows:

- MSMM_RNGOFFHK Solicited off-hook detected
- MSMM_TERM Ringing terminated

In synchronous mode, AT_FAILURE indicates failure and a positive value (>0) indicates the reason for termination. Reasons for termination are:

- MSMM_RNGOFFHK Solicited off-hook detected
- MSMM_TERM Ringing terminated

■ Cautions

This function fails when:

- The board is not an MSI/SC-R board.
- The device handle is invalid.

■ Example

Synchronous mode:

```
#include <windows.h>
#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 MSI B1, station 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);
}
```

Asynchronous mode:

```
#include <windows.h>
#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 MSI B1, station 1: errno=%d", errno);
    exit(1);
}

/* Set SRL to run in polled mode */
srlmode = SR_POLLMODE;
```

```
if (sr_setparm(SRL_DEVICE,SR_MODEID, (void *)&srlmode) == -1) {
    /* process error */
}

/* 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);
}

/* 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
    */
}
```

Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

See Also

- **ms_setevtmsk( )**

Name:	int ms_genziptone(devh)	
Inputs:	int devh	• MSI/SC station device handle
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Station	
Mode:	synchronous	

■ Description

The **ms_genziptone( )** function generates a zip tone to the station associated with the device handle. 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/SC station that is not part of a conference or routed to an SCbus time slot.

Parameter	Description
devh:	The valid MSI/SC 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.

■ Example

```
#include <windows.h>
#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);
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

Name:	int ms_getbrdparm(devh,param,valuep)	
Inputs:	int devh	• MSI/SC 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 AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

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

Parameter	Description
devh:	The valid device handle returned by a call to **ms_open( )**.
param:	The parameter to be examined.
valuep:	Pointer to the variable where the parameter value will be returned.

■ Cautions

This function fails when:

- The device handle is invalid.
- The parameter specified is invalid.

■ Example

```
#include <windows.h>
#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;
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

See Also

- **ms_setbrdparm( )**

Name:	int ms_getcde(devh,confID,cdt)	
Inputs:	int devh	• MSI/SC board device handle
	int confID	• conference identifier
	MS_CDT *cdt	• pointer to MS_CDT structure
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ Description

The **ms_getcde( )** function retrieves the attributes of a conferee in an existing conference.

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier.
cdt:	Pointer to an MS_CDT structure.

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

The MS_CDT structure has the following format:

```
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;
```

The chan_num denotes the station number or the SCbus time slot number of the device. The chan_sel defines the meaning of chan_num. Valid choices are as follows:

- MSPN_STATION MSI/SC station number

- MSPN_TS SCbus time slot number

The chan_attr is a bitmask describes the party's properties within the conference. Possible returns are listed in the table below. It is possible that a combination of any of the attributes shown in the table will be returned.

Table 8. Possible Returns for Channel Attribute

Channel Attribute	Description
MSPA_NULL	No special attributes for party.
MSPA_RO	Party participates in conference in receive-only mode.
MSPA_TARIFF	Party receives periodic tone for duration of call.
MSPA_COACH	Party is a coach. Coach is heard by pupil only.
MSPA_PUPIL	Party is a pupil. Pupil hears everyone including coach.
MSPA_NOAGC	Disabled automatic gain control.

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.

■ Example

```
#include <windows.h>
#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
 *
 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

See Also

- **ms_setcde( )**

Name:	int ms_getcnflist(devh,confID,numpty,cdt)	
Inputs:	int devh	• MSI/SC 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 AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Attribute	
Mode:	synchronous	

■ 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/ SCbus time slot number, selector, and attribute description.

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

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier.
numpty:	Pointer to the party count.
cdt:	Pointer to the conference descriptor table.

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.

The conference descriptor table is an array of the MS_CDT structure. The MS_CDT structure has the following format:

```
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;
```

■ Cautions

This function fails when an invalid conference ID is specified.

■ Example

```
#include <windows.h>
#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);
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to

obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

See Also

- **ms_estconf( )**

Name: int ms_getctinfo (devh,ct_devinfop)

Inputs: int devh • MSI/SC station device handle
CT_DEVINFO *ct_devinfop • pointer to information structure

Returns: 0 on success
AT_FAILURE on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Attribute

Mode: synchronous

■ Description

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

Parameter	Description
devh:	The station device handle.
ct_devinfop:	Pointer to the channel/station information structure.

On return from the function, the CT_DEVINFO structure contains the relevant information. The CT_DEVINFO structure is declared as follows:-

```
typedef struct {
    unsigned long   ct_prodid;              /* Dialogic product ID */
    unsigned char   ct_devfamily;           /* Device family */
    unsigned char   ct_devmode;             /* Device mode */
    unsigned char   ct_nettype;             /* Network device type */
    unsigned char   ct_busmode;             /* Bus mode (PEB/SCbus) */
    unsigned char   ct_busencoding;         /* PCM encoding (Mu-Law/A-Law) */
    unsigned char   ct_rfu[7];              /* reserved for future use */
} CT_DEVINFO;
```

The valid choices for each member of the CT_DEVINFO structure are defined in *dtilib.h*.

The ct_prodid field contains a valid Dialogic product identification number for the device. The MSI/SC board's product ID is found in the *msilib.h* file.

The ct_devfamily specifies the device family and will contain the following:

CT_DFMSI MSI/SC station device

NOTE: The device mode (ct_devmode) field is not relevant for the MSI/SC board.

The ct_nettype member of the CT_DEVINFO structure contains the following for the MSI/SC:

CT_NTSTATION Station front end

The ct_busmode specifies the bus architecture the device uses to communicate with other devices in the system.

CT_BMSCBUS Signal Computing System Architecture bus (SCbus)

The ct_busencoding field describes the PCM encoding being used on the bus. Valid choices are:

CT_BEULAW Mu-law encoding

CT_BEALAW A-law encoding

■ Cautions

This function fails if an invalid station handle is specified.

■ Example

```
#include <windows.h>
#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);
```

■ Errors

If the function does not complete successfully, it will return AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name:	int ms_getevt(devh,eblkp,timeout)	
Inputs:	int devh	• MSI/SC device handle
	EV_EBLK * eblkp	• pointer to event block
	int timeout	• timeout value
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **ms_getevt( )** function blocks and returns control to the application. 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 **timeout** 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:	Specifies the maximum amount of time to wait for an event to occur. If timeout is set to -1, the **ms_getevt( )** function does not timeout and blocks until an event occurs. If timeout is set to 0 and an event is not present, the function returns immediately with a AT_FAILURE return code.

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

eblk.ev_dev	The device on which the event occurred. This is the same as the **devh** passed to the function.
eblk.ev_event	MSEV_SIGEVT indicating signaling transition event.
eblk.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

This function fails when:

- The device handle is invalid for an MSI/SC device.
- The event field is invalid.

■ 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, AT_FAILURE );    /* 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 */

}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_getevtmsk( )**

Name:	int ms_getevtmsk(devh,event,bitmaskp)	
Inputs:	int devh	• MSI/SC station device handle
	int event	• event to retrieve
	unsigned short *bitmaskp	• pointer to bitmask variable
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **ms_getevtmsk( )** function returns 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 On-hook transition event Off-hook transition event Hookflash event
bitmaskp:	Points 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/SC station device.
- The event field is invalid.

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      printf( "Cannot open board msiB1C1. errno = %d", errno );
      exit( 1 );
   }
   /* Get signaling event mask*/
   if ( ms_getevtmsk( tsdev, MSEV_SIGMSK, &bitmask ) == AT_FAILURE ) {
      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 ) == AT_FAILURE ) {
      printf( "Cannot close board msiB1C1. errno = %d", errno );
   }
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_setevtmsk( )**

Name:	int ms_monconf(devh,confID,lts)	
Inputs:	int devh	• MSI/SC board device handle
	int confID	• conference identifier
	long *lts	• pointer to listen SCbus time slot
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ Description

The **ms_monconf( )** function adds a monitor to a conference. Monitoring a conference guarantees that the conferenced signal will be placed on the SCbus. 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 SCbus, depending on the chan_sel of the party.

Since the monitored signal is on the SCbus, several parties can listen to the monitored signal simultaneously.

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier.
lts:	Pointer to the listen SCbus 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.

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.

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

■ 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.

■ Example

```
#include <windows.h>
#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_tsarray = &lts;
if (dt_listen(tsdev1,&tsinfo) == -1){
    printf("Error Message = %s",ATDV_ERRMSGP(tsdev1));
    exit(1);
}
/*
 * Continue Processing
 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_unmonconf( )**

Name: int ms_open(name,oflags)
Inputs: char *name
- MSI/SC station or board device name
- open attribute flags

int oflags
Returns: device handle
AT_FAILURE on failure
Includes: srllib.h
dtilib.h
msilib.h
Category: Device Management
Mode: synchronous

■ Description

The **ms_open( )** function opens an MSI/SC 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/SC 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 the Windows NT **open( )**.

This function fails when:

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

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      printf( "Cannot open board msiB1. errno = %d\n", errno );
      exit( 1 );
   }

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

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

}
```

■ 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 AT_FAILURE 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.

See Also

- **ms_close( )**

Name: int ms_remfromconf(devh,confID,cdt)

Inputs: int devh
- MSI/SC board device handle

int confID
- conference identifier

MS_CDT *cdt
- pointer to MS_CDT structure

Returns: 0 on success
AT_FAILURE on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Conference Management

Mode: synchronous

■ 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. For a full description of the MS_CDT structure, see the **ms_estconf( )** function.

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier number.
cdt:	Pointer to an MS_CDT structure.

NOTES: **1.** Dialogic recommends that you unlisten before removing the SCbus 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.

■ Example

```
#include <windows.h>
#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 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

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

Name: int ms_setbrdparm(devh,param, valuep)

Inputs:
int devh — • MSI/SC board device handle
unsigned long param — • device parameter defined name
void * valuep — • pointer to parameter value

Returns: 0 on success
AT_FAILURE on failure

Includes: srllib.h
dtilib.h
msilib.h

Category: Configuration

Mode: synchronous

■ Description

The **ms_setbrdparm( )** function changes board parameters.

Parameter	Description
devh:	The valid board device handle returned by a call to **ms_open( )**.
param:	The parameter whose value is to be altered.
valuep:	Void pointer to location containing the parameter value.

NOTE: Calling this function may cause the available resource count to change. 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.

Typically, the default value for each MSI/SC parameter is adequate for operation. However, the user may need to change the following conditions:

- MSG_DBONTM — Debounce on time
- MSG_DBOFFTM — Debounce off time
- MSG_MINFLASH — Minimum hook flash time
- MSG_MAXFLASH — Maximum hook flash time

- MSG_PDRNGCAD — Select predefined ring cadence pattern
- MSG_RING — Ringing capability support information
- MSG_RNGCAD — Ring cadence pattern
- MSG_UDRNGCAD — Set user-defined ring cadence pattern
- MSG_ZIPENA — Ziptone enable
- MSCB_ND — Notify on add
- MSCB_ZIP — Zip tone notification

The following table contains a description of MSI/SC device parameters:

Table 9. MSI/SC Board/Device Parameters

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. The MS_NCB structure is as follows:
	`typedef struct ms_ncb{`
	`  unsigned char volume;  /* volume */`
	`  unsigned char tone;  /* tone frequency */`
	`  short duration;  /* tone duration */`
	`  short pulse;  /* pulse repetition interval */`
	`} MS_NCB`
	
	Defaults: volume =7, frequency = 24H or 1125 Hz, duration = 14H or 200 ms.

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 block are set

Parameter ID	Description
	but the pulse repetition field is ignored by the function. Defaults: Volume = 7, frequency = 18H, duration = 64H or 1sec.
MSG_DBOFFTM	Defines the minimum length of time (50 ms units) before an off-hook transition is detected. Off-hook debounce time range: 2-14H, default = 3H. A pointer to a short containing this duration is passed as the third parameter.
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: 15H. A pointer to a short containing this duration is passed as the third parameter. **NOTE:** The MSG_DBONTM 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.
MSG_MAXFLASH	Defines a maximum length of time for a station to be in an on-hook state before a hook flash signal is detected. Maximum hook flash time range: 4-3CH, default = 14H. A pointer to a short containing this duration is passed as the third parameter.
MSG_MINFLASH	Defines a minimum length of time 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.
MSG_PDRNGCAD	This parameter is used to select one of the following predefined ring cadence patterns on the MSI/SC board (duration in seconds): **Value** **Cadence Pattern**

Parameter ID	Description	
	1	1 on 5 off
	2	1 on 2.75 off
	3	1.5 on 3 off
	4	1 on 4.25 off
	5	.5 on, 2.5 off .5 on, 2.5 off
	6	2 on 4 off
	The default is value 6.	
MSG_RING	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_NONRNGBRD**.	
MSG_RNGCAD	This parameter is used to get the ring cadence pattern. The length, in bytes, of this parameter 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).	
MSG_UDRNGCAD	Specifies the user-defined ring cadence. This parameter is used to set the station ring cadence (the repeating pattern of ringing ON/OFF durations) for all stations attached to the MSI/SC board.	

Parameter ID	Description
MSG_ZIPENA	The zip tone setting. MS_ZIPENABLE enables zip tone generation. MS_ZIPDISABLE disables zip tone generation. Default = MS_ZIPENABLE.

The ring cadence is 1/3 active and 2/3 inactive. The active period defines an ON/OFF pattern of ringing in units of 250ms and is specified in the value pointed to by **valuep**. The value can be from 2 to 7 bytes, depending on the duration of the active period.

Byte 1 specifies the total number of bits in the active period, ranging from 01H to 30H (1-48 bits). Since each bit represents a 250 ms duration, the active period can range from 250 ms to 12 seconds.

Bytes 2-7 (the number of bytes depends upon the value specified in Byte 1) specifies the active period ring pattern. Each bit represents the state of the ring current (1=ON, 0=OFF) for a 250 ms duration in a sequence from left to right.

The inactive period is a mandatory time of no ringing that is twice the active period duration. An inactive periods can range from 500 ms to 24 seconds and is created from the active period duration. The default ring cadence is 2 seconds on and 4 seconds off.

Table 10. MSI/SC Ring Cadence Examples

Example Number	Desired Cadence (seconds) Ring ON Time (embedded off time)	Parameter Value (hexadecimal) Ring OFF Time	Total Bits (byte 1)	Active Pattern (bytes 2-n)
Single Ring Patterns:				
1	.75	7.5	0B	E000
2	1	2	04	F0
3	1	2.75	05	F0
4	1	4.25	07	F0
5	1	5	08	F0
6	1.25	4.75	08	F8
7	1.5	3	06	FC
8	1.5	3.75	07	FC
9	2	4	08	FF
Double Ring Patterns:				
10	.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	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

■ 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.

■ Example

```
#include <windows.h>
#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);
   }
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_getbrdparm( )**

Name:	int ms_setcde(devh,confID,cdt)	
Inputs:	int devh	• MSI/SC device handle
	int confID	• conference identifier
	MS_CDT *cdt	• pointer to an MS_CDT structure
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ Description

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

Parameter	Description
devh:	The MSI/SC device handle.
confID:	The conference identifier number.
cdt:	Pointer to an MS_CDT structure.

The MS_CDT structure has the following format:

```
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;
```

The chan_num denotes the station number or the SCbus time slot number of the party in the conference. The chan_sel defines the meaning of the chan_num. Valid choices are as follows:

- MSPN_STATION — MSI/SC station number
- MSPN_TS — SCbus time slot number

Channel attribute is a bitmask describing the party's properties within the conference. It will return one or more of the following values ORed together:

- MSPA_NULL — No special attributes.
- MSPA_RO — Party participates in conference in receive-only mode.
- MSPA_TARIFF — Party receives periodic tone for duration of call.
- MSPA_COACH — Party is a coach. Coach is heard by pupil only.
- MSPA_PUPIL — Party is a pupil. Pupil hears everyone including coach.

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

■ Cautions

This function fails when:

- The device handle specified is invalid.
- The device is not connected to the SCbus.

■ Example

```
#include <windows.h>
#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
 *
 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

See Also

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

Name:	ms_setevtmsk(devh,event,bitmask,action)	
Inputs:	int devh	• MSI/SC 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 AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Configuration	
Mode:	synchronous	

■ 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:	One 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 (see below).
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 the DTA_SETMSK bitmask 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_SIG 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:

1. Call **sr_enbhdlr( )**. This function specifies the event and the application defined event handler that is called from a signal handler.
2. Call **ms_setevtmsk( )**. This function specifies the list of events the application should be notified of.

   NOTE: For an event to be handled, it must be specified in both **sr_enbhdlr( )** and **ms_setevtmsk( )**.

3. The event data is retrieved using the **sr_getevtdatap( )** function. Refer to *Appendix A* for more information.

■ Cautions

This function fails when:

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

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      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 ) == AT_FAILURE ){
      do_error( tsdev, "ms_setevtmsk()" );
      exit( 1 );
   }

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

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

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_getevtmsk( )**

Name:	ms_setstparm(devh,param,valuep)	
Inputs:	int devh	• MSI/SC station device handle
	unsigned char param	• parameter name
	void *valuep	• pointer to parameter value
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **ms_setstparm( )** function changes the MSI/SC station level parameters.

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.

■ Cautions

This function fails when:

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

■ Example

```
#include <windows.h>
#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);
   }
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_getbrdparm( )**

- **ms_setbrdparm( )**

Name:	int ms_setvol (devh,type,steps)	
Inputs:	int devh	• MSI/SC station device handle
	int type	• volume adjust or reset
	int steps	• number of steps to increase or decrease volume
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Station	
Mode:	synchronous	

■ 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, each **step** increases or decreases from the current volume by 1 dB. A positive **step** value increases the volume, and a negative **step** 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/SC board.

■ Example

```
#include <windows.h>
#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
 *
 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

Name:	int ms_stopfn(devh,funcid)	
Inputs:	unsigned int devh	• MSI/SC station device handle
	unsigned int funcid	• ID of multitasking function
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Device Management	
Mode:	synchronous	

■ Description

The **ms_stopfn( )** function stops a multitasking function in progress for a station.

Parameter	Description
devh:	The MSI/SC 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.

NOTE: Currently, only ringing can be stopped using **ms_stopfn( )**. The ringing must have been started by issuing a **ms_genring( )** for that station.

■ Cautions

This function fails when the device is not an MSI/SC station.

■ Example

```
#include <windows.h>
#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);
}
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_genring( )**

Name:	int ms_tstcom(devh,tmo)	
Inputs:	int devh	• MSI/SC board device handle
	int tmo	• timeout value
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Diagnostic	
Mode:	synchronous/asynchronous	

■ 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.

■ Synchronous Mode

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

■ Asynchronous Mode

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 *Appendix A* for information on 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).

■ Example

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      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 ) == AT_FAILURE ) {
      do_error( bddev, "ms_tstcom()" );
      exit( 1 );
   }

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

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

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

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_tstdat( )**

Name:	int ms_tstdat(devh,tmo)	
Inputs:	int devh	• MSI/SC board device handle
	int tmo	• timeout value
Returns:	0 on success AT_FAILURE on failure	
Includes:	srllib.h dtilib.h msilib.h	
Category:	Diagnostic	
Mode:	synchronous/asynchronous	

■ Description

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

■ Asynchronous Mode

To operate this function in (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 *Appendix A* for information on event handling.

■ Synchronous Mode

To run this function in (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.

■ Cautions

This is a board level function only.

This function fails when:

- The test data is corrupted.
- The device handle is invalid.

■ Example

Synchronous mode:

```
#include <windows.h>
#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 ) ) == AT_FAILURE ) {
      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 ) == AT_FAILURE ) {
      do_error( bddev, "ms_tstdat()" );
```

```
      exit( 1 );
   }

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

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

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

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_tstcom( )**

Name:	int ms_unmonconf(devh,confID)	
Inputs:	int devh	• MSI/SC board device handle
	int confID	• conference ID
Returns:	0 on success	
	AT_FAILURE on failure	
Includes:	srllib.h	
	dtilib.h	
	msilib.h	
Category:	Conference Management	
Mode:	synchronous	

■ Description

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

NOTE: Calling this function frees one resource.

Parameter	Description
devh:	The MSI/SC board device handle.
confID:	The conference identifier.

■ Cautions

This function fails when:

- The device handle specified is invalid.
- An invalid conference is specified.
- A monitor does not exist in the conference.

■ Example

```
#include <windows.h>
#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_tsarray = &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 */
```

■ Errors

If the function does not complete successfully, it will return a AT_FAILURE to indicate error. Use the Standard Attribute function **ATDV_LASTERR( )** to obtain the applicable error value(s). Refer to the error type tables found in *Chapter 2. MSI/SC Library Function Overview* of this guide.

Error defines can be found in *dtilib.h* or *msilib.h*.

■ See Also

- **ms_estconf( )**
- **ms_monconf( )**

4. MSI/SC Application Guidelines

4.1. General Guidelines

This chapter contains suggestions to guide programmers in designing and coding a Dialogic MSI/SC application for Windows NT.

This section provides the following MSI/SC general and task-specific programming guidelines:

- General Guidelines
- Initialization
- Compiling and Linking
- Aborting

The following general guidelines for writing Dialogic applications are explained in this section:

NOTE: These guidelines are not a comprehensive guide to developing or debugging MSI/SC applications.

- Using symbolic defines
- Including header files
- Checking return codes

4.1.1. Use Symbolic Defines

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

4.1.2. Include Header Files

Various header files must be included in your application to test for error conditions, to use library functions from other Dialogic products, or to perform

event-management and standard-attribute functions. An example is shown below. See *2. MSI/SC Library Function Overview* for details.

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

NOTE: To avoid redundancy in the remaining programming examples in this chapter, **#include** statements will not be shown.

4.1.3. Check Return Codes

Most Network and MSI/SC Windows NT library functions return a value of AT_FAILURE if they fail (extended attribute functions return AT_FAILURE or AT_FAILUREP). Therefore, any call to a library function should check for a return value indicating an error. This can be done by using a format similar to the following:

```
/* call to Dialogic MSI/SC library function */

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

Using this technique ensures that all errors resulting from a library call will be trapped and handled properly by the application. In many cases, you can check for a return value of other than zero (0), as shown in the example below. However, this should only be used where a nonzero value is returned when the function fails. For details, see *2. MSI/SC Library Function Overview* and *3. MSI/SC Function Reference*.

```
/* error handling routine */
void do_error( devh, funcname )

int devh;
    char *funcname;
{
   int errorval = ATDV_LASTERR( devh );
   printf( "Error while calling function %s on device %s.  \n", funcname,
      ATDV_NAMEP( devh ) );
   if ( errorval == E_MSSYSTEM ) {
       printf( "errno = %d\n", errno );
       perror("");
   } else {
       printf( "Error value = %d\n Error message = %s\n",
```

```
         errorval,ATDV_ERRMSGP( devh ) );
   }
   return;
}
main( )
{
         .
         .
         .
   /* call to Dialogic MSI/SC library function */
   if (ms_setevtmsk( devh, MSEV_SIGMSK, 0, DTA_SETMSK ) ) != 0) {
      do_error( devh, "ms_setevtmsk()" );
   }
   /* successful function call -
      continue processing ...  */
            .
            .
            .
}
```

NOTES:

1. Calls to **ms_open( )** return either AT_FAILURE or a nonzero device handle. Therefore, when issuing the **ms_open( )** function, check for a return of -1. The specific error can be found in the global variable **errno**, contained in *errno.h*.
2. Calls to **ATMS_TSSGBIT( )** return the pointer AT_FAILUREP when the function fails.
3. To avoid redundancy in the remaining programming examples in this chapter, the **do_error( )** function will not be shown.

4.2. Initialization

Before an MSI/SC application can perform any processing or access devices, it should initialize the MSI/SC hardware to correspond with the physical configuration of your system and set other parameters needed to support the application. Tasks that are performed as a part of initialization generally include:

- Set hardware configuration
- Set event masks
- Initialize stations

These involve the following MSI/SC Windows NT functions:

- **ms_setevtmsk( )**
- **ms_setbrdparm( )**

4.2.1. Set Hardware Configuration

Use **ms_setbrdparm( )** to set hardware configuration, debounce times, minimum and maximum hook flash times,. ring cadence patterns, and ziptone. Specific settings include:

• MSG_DBONTM	Debounce on time
• MSG_DBOFFTM	Debounce off time
• MSG_MINFLASH	Minimum hook flash time
• MSG_MAXFLASH	Maximum hook flash time
• MSG_PDRNGCAD	Select the predefined ring cadence pattern
• MSG_RING	Ringing capability support information
• MSG_RNGCAD	Ring cadence pattern
• MSG_UDRNGCAD	Set user-defined ring cadence pattern
• MSG_ZIPENA	Ziptone enable
• MSCB_ND	Notify on add
• MSCB_ZIP	Zip tone notification

4.2.2. Set event mask on MSI/SC stations

Use the **ms_setevtmsk( )** function for setting and clearing the mask for on-hook transitions, off-hook transitions, and hook flash detection. The MS_OFFHOOK, MS_ONHOOK, and MS_HOOKFLASH equates are used for setting and clearing respective bits in the message mask.

4.2.3. Terminating

When your process completes, devices should be shut down in an orderly fashion. Tasks that are performed to terminate an application generally include:

- Disabling events
- Resetting time slots
- Closing devices

The **ms_setevtmsk( )** function can disable all currently enabled event notification masks.

NOTE: SRL Event Management functions such as **sr_dishdlr( )**, (which disables an event handler), must be called before closing the device that is sending the handler event notifications (see *Appendix A* for SRL details).

4.3. Compiling and Linking

To compile and link your application, follow the syntax instructions for your version of the Windows NT C Development Package.

Include the following libraries when using the MSI/SC:

```
libsrlnt.lib
libdtint.lib
```

Depending on your application, you may need to link with other libraries. Refer to the appropriate documentation for other Dialogic products.

4.4. Aborting

If you abort an MSI/SC Windows NT application by pressing the interrupt key, the Windows NT system will terminate the current process but may leave devices in an unknown state. The next time you run your application, therefore, you may encounter errors.

To avoid errors of this type, your application should include an event handler that traps the interrupt key and performs the actions listed in *Section 4.2.3. Terminating*.

Appendix A

Standard Runtime Library: MSI/SC Entries and Returns

The Standard Runtime Library is a device-independent library containing Event Management functions, Standard Attribute functions, and the DV_TPT Termination Parameter table. SRL functions and data structures are described in detail in the *Standard Runtime Library Programmer's Guide* and the *Voice Software Reference for Windows NT*.

This appendix lists the MSI/SC entries and returns for each of the Standard Runtime Library (SRL) components.

Table 11. Guide to Appendix A

SRL Component	MSI/SC Data Listed in Appendix A
Event Management functions	MSI/SC inputs for Event Management functions.
	MSI/SC returns from Event Management functions.
Standard Attribute functions	MSI/SC values returned by the Standard Attribute functions.
DV_TPT table	Termination conditions and related data. Not used by the MSI/SC device.

Event Management Functions

The Event Management functions retrieve and handle MSI/SC termination events for the **ms_setevtmsk( )**, **ms_tstcom( )**, and **ms_tstdat( )** functions.

The Event Management functions are listed in the following tables.

Table 12. MSI/SC Inputs for Event Management Functions

Event Management Function	MSI/SC specific Input	Value
sr_enbhdlr( )	event type	MSEV_NORING

Event Management Function	MSI/SC specific Input	Value
Enable event handler		MSEV_RING MSEV_SIGEVT DTEV_COMRSP DTEV_DATRSP
sr_dishdlr( ) Disable event handler	event type	Same as above.
sr_waitevt( ) Wait for next event	N/A	N/A
sr_waitevtEX( ) Extended wait event	list of device handles	N/A

Table 13. MSI/SC Returns from Event Management Functions

Event Management Function	MSI/SC specific Input	Value
sr_getevtdev( ) Get Dialogic device handle	device	MSI/SC device handle.
sr_getevttype( ) Get event type	event type	MSEV_NORING MSEV_RING MSEV_SIGEVT DTEV_COMRSP DTEV_DATRSP
sr_getevtlen( ) Get event length	event length	Number of bytes in the data returned.
sr_getevtdatap( ) Get pointer to event data	event data	Pointer to variable containing the value of a selected bitmask.

Standard Attribute Functions

Standard Attribute functions return general device information such as the device name, or the last error that occurred on the device. The Standard Attribute functions and the MSI/SC-specific information returned are listed below.

Table 14. Standard Attribute Functions

Standard Attribute Function	Information Returned for MSI/SC
ATDV_ERRMSGP( )	Pointer to string describing the error that occurred during the last function call on a device.
ATDV_IRQNUM( )	Interrupt number for a specified device.
ATDV_LASTERR( )	The error that occurred during the last function call on a specified device.
ATDV_NAMEP( )	Pointer to device name (MSIBb).
ATDV_SUBDEVS( )	Number of channels. The MSI/240SC board has 24 channels. Refer to the *Standard Runtime Library Programmer's Guide* for information on subdevices.

Appendix B

Related MSI/SC Publications

Below is a list of Dialogic publications to read for information on products related to the MSI/SC.

Dialogic References

- *Digital Network Interface Software Reference for Windows NT*
- MSI/SC *Quick Install Card*
- *Voice Software Reference for Windows NT*
- *System Release Software Installation Reference for Windows NT*
- *SCbus Routing Guide*
- *SCbus Routing Function Reference for Windows NT*

Dialogic Application Notes

- *Converting a Windows NT Application from PEB to SCbus*

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/SC board.

A-Law: A pulse-code modulation (PCM) algorithm used in digitizing telephone audio signals in E-1 areas.

analog: In this guide, analog refers to agent communications between a headset and the MSI/SC or to the *loop-start* type of network interface.

asynchronous function: Allows program execution to continue without waiting for a task to complete. See *synchronous function*.

automatic call distribution: 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 a 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 E-1 frame. (See *time slot*.) 3. When used in reference to a bus, an electrical circuit carrying control information and data.

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/SC daughterboard assembly. The daughterboard enables the MSI/SC 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 semipermanent way.

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.

E-1: Another name given to the CEPT digital telephony format devised by the CCITT that carries data at the rate of 2.048 Mbps (DS-1level). 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 Dialogic device handle) and return device-specific information.

full-duplex: Transmission in two directions simultaneously, or more technically, bidirectional, simultaneous two-way communications.

host PC: The system PC in which 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 (T-1 areas).

MSI: Modular Station Interface. A PEB-based Dialogic expansion board that interfaces PEB time slots to analog station devices by way of modular daughterboards.

MSI/SC: Modular Station Interface. An SCbus-based Dialogic expansion board that interfaces SCbus time slots to analog station devices.

PC: Personal computer. In this guide, the term refers to an IBM Personal Computer or compatible machine.

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.

PEB: PCM Expansion Bus. The common communication medium for passing signaling, audio, and control information between Dialogic and other PEB-compatible expansion boards. Information on the PEB is encoded differently depending on the telephony standard implemented by board hardware and firmware.

rfu: Reserved for future use.

SCbus: Signal Computing bus. A hardwired connection between Switch Handlers (SC2000 chips) on SCbus-based products for transmitting information over 1024 time slots to all devices connected to the SCbus.

SCbus routing functions: Setup communications between devices connected to the SCbus. These functions enable an application to connect or disconnect (make or break) the receive (listen) channel of a device to or from an SCbus time slot.

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*.

SpringBoard: A Dialogic expansion board using digital signal processing to emulate the functions of other products. The SpringBoard is a development platform for Dialogic products such as the D/121B.

SRL: Standard Runtime Library containing Event Management functions, Standard Attribute functions, and data structures that are used by all Dialogic devices.

Standard Attribute functions: Class of functions that take one input parameter (a valid Dialogic device handle) and return generic information about the device. For instance, Standard Attribute functions return IRQ and error information for all device types.The Dialogic SRL contains Standard Attribute functions for all 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. See *asynchronous function*.

T-1: The digital telephony format used in North America and Japan that carries data at the rate of 1.544 Mbps (DS-1 level).

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.

ziptone: Short burst of a specified tone to an ACD agent headset usually indicating a call is being connected to the agent console.

Index

NOTES

NOTES

NOTES