Dialogic®

Dialogic® Station Side Interface API

Library Reference

April 2008

05-2462-003

Publication Date: April 2008

Document Number: 05-2462-003

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2462-003	April 2008	Made global changes to reflect Dialogic brand.
05-2462-002	November 2005	Initial version of document for Dialogic® Host Media Processing Software 2.0 for Windows® Operating Systems.
05-2462-001	June 2005	Preliminary version of document for Dialogic® Media Exchange Architecture release.

Table of Contents

1 Introduction

Although the Dialogic® Modular Station Interface (MSI) API provides station-side (phone-driving) application interfaces to FXS hardware, proprietary digital station sets (business phones) offer a set of low-level controllable features that are not supported by the MSI API. These features include station text display control, station indicator (lamps) control, multiple ringer cadence, and tone control. The Dialogic® Station-Side Interface (SI) API is designed to support these features of digital station sets.

The Dialogic® SI API is a low-level station-side API providing a superset of station-side control capabilities. The SI API supports proprietary digital station sets as well as station-side implementations (analog, Megaco device, etc.). The SI API can be used with Dialogic® Host Media Processing (HMP) Software PBX Enabled Boards and provides call control processing for phone driving boards and station-side interfaces for proprietary digital station sets.

2 Description

Devices that implement the SI API must support the virtual device type of "ssi". Therefore, a board that supports the SI API must map to the board name "ssiBx" and a channel (station) must map to the station device name "ssiBxCy".

All API functions that interface with external devices or hardware to complete the function request support both synchronous completion and asynchronous completion.

The SI API can be used with Dialogic® HMP Software PBX Enabled Boards and provides call control processing for phone driving boards and station-side application programming interfaces for proprietary digital station sets. In this configuration, the SI API manages the retrieval of events via the Standard Runtime Library (SRL) API.

The SI API is also supported in a stand-alone mode (SA Mode) independent of system release or Dialogic HMP Software. When operating in this configuration, the SI API does not rely on the SRL API, but manages the retrieval of events via **si_WaitEvent( )**.

API Description

3.1 Device Open & Close

3.1.1 Functions

```
int si_Open(int* piHandle, const char* szDevName, TEnumSyncMode eSyncMode,
const void* pvUserContext)
```

Inputs

int*	piHandle	- place to write station handle
const char*	szDevName	- ssi station device name
int	eSyncMode	- Enum_SyncMode, Enum_AsyncMode
const void*	pvUserContext	- User-supplied pointer to match event received in

Enum_AsyncMode mode with async function calls

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Opens the specified station device. Returns opened station handle in piHandle. Station device is in the format of “ssiB1C1”.

When using **si_Open( )** in synchronous mode, a device handle is returned in piHandle only if the function is successful (returns SI_SUCCESS).

When using **si_Open( )** in asynchronous mode, a device handle is returned in piHandle upon return from the **si_Open**() function call so that the termination event may be matched to the device handle. If the asynchronous open fails with a termination event of Enum_SiEvOpenFail, the application must use **si_Close()** on the device handle to free resources.

Errors

If this function is called synchronously, but fails with a returns of SI_FAILURE(-1), no device handle is created. Therefore, ATDV_LASTERR() cannot be used to obtain an error code.

If this function is called asynchronously, but subsequently fails with an Enum_SiEvOpenFail termination event, the application may call ATDV_LASTERR() to obtain the error code before calling si_Close() on the returned device handle to free resources.

Enum_SiErrorSystem	Internal error creating resources for the station.

Termination Event

Enum_SiEvOpenSuccess
Enum_SiEvOpenFail

int si_Close(int iHandle)

Inputs

int iHandle - station handle

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Closes the specified station device.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

3.1.2 Example

#include “silib.h”

```
int g_stationHandle;
void main()
{
      if ((si_Open(&g_stationHandle, “ssiB1C1”, Enum_SyncMode, NULL)) == -1)

            printf(“Failed to open ssiB1C1\n”);
            return;
      }
      else
      {
            printf(“Station Handle: %d\n”, g_stationHandle);
      }

      if( (si_Close(g_stationHandle)) == -1)
      {
            printf(“Failed to close %s\n”,
                  ATDV_NAMEP(g_stationHandle));
            printf(“Error code = 0x%x, Error message = %s\n”,\
                  ATDV_LASTERR(g_stationHandle), ATDV_ERRMSGP(g_stationHandle));
      }
      return;
}
```

3.2 Capabilities Determination

3.2.1 Functions

```
int si_GetBoardCount(int* piNumBoards, TEnumSiError* peError)
```

Inputs

int*	piNumBoards	- returns number of “ssi” boards
TEnumSiError*	peError	- if SI_FAILURE return, error code returned here

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the number of "ssi" boards in the system.

Errors

Enum_SiErrorSysNotStarted	System was not started correctly.
Enum_SiErrorBadVal	Required pointer is invalid.

```
int si_GetBoardName(int iBoardIndex, char* pszBoardName, int*
piBoardNameLen, TEnumSiError* peError)
```

Inputs

int	iBoardIndex	- 0-based index of the board
char*	pszBoardName	- buffer in which to return board name
int*	piBoardNameLen	- length of buffer pointed to by pszBoardName
TEnumSiError*	peError	- if SI_FAILURE return, error code returned here

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Board name of the specified board index. On input, piBoardNameLen must point to length of buffer pszBoardName. If buffer length is insufficient (Enum_SiErrorInvLen error code returned in peError), the required buffer length is returned in piBoardNameLen.

Errors

Enum_SiErrorSysNotStarted	System was not started correctly.
Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvBd	Board index is out of range.
Enum_SiErrorInvLen	Specified length is too small.

```
int si_GetChannelCount(const char* szBoardName, int* piNumChannels,
TEnumSiError* peError)
```

Inputs

const char*	szBoardName	- ssi board name
int*	piNumChannels	- returns number of channels on board
TEnumSiError*	peError	- if SI_FAILURE return, error code returned here

Returns

Number of stations supported by the specified board.

Description

Retrieves the number of "ssi" stations supported by the specified board. For example, “ssiB1”.

Errors

Enum_SiErrorSysNotStarted	System was not started correctly.
Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvName	Board name is invalid.

```
int si_GetStationCapabilities(const char* szDevName, TEnumSiCapabilities
eCapability, void* pCapBlk, int iLen, TEnumSiError* peError)
```

Inputs

const char*	szDevName	- ssi board name
TEnumSiCapabilities	eCapability	- capability to retrieve
void*	pCapBlk	- capabilities block
int	iLen	- length of buffer pointed to by pCapBlk
TEnumSiError*	peError	- if SI_FAILURE return, error code returned here

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the capabilities of the specified station.

Errors

Enum_SiErrorSysNotStarted	System was not started correctly.
Enum_SiErrorBadVal	Unrecognized capability or a required pointer is invalid.
Enum_SiErrorInvName	Board name is invalid.
Enum_SiErrorBadBrd	Board is in a bad state.
Enum_SiErrorInvLen	Specified length is not equal to the size of the requested capabilities structure.

3.2.2 Types

```
enum
{
        Enum_SiCapabilitiesInvalid = 0,
        Enum_SiCapabilitiesMain,                    /* retrieves TSSiCapabilitiesMain */
        Enum_SiCapabilitiesKeys,                    /* retrieves array of TSSiCapabilitiesKey */
        Enum_SiCapabilitiesIndicators,              /* retrieves array of TSSiCapabilitiesIndicator */
        Enum_SiCapabilitiesVolumeDevices,           /* retrieves array of TSSiCapabilitiesVolumeDevice */
        Enum_SiCapabilitiesSensitivityDevices,      /* retrieves array of TSSiCapabilitiesSensitivityDevice */
        Enum_SiCapabilitiesAudioDevices,            /* retrieves array of TSSiCapabilitiesAudioDevice */
        Enum_SiCapabilitiesSoftKeys                 /* retrieves array of TSSiCapabilitiesSoftKey */
} TEnumSiCapabilities;
```

```
typedef struct SSiCapabilitiesMain
{
char                          szType[128];          /* station type */
char                          szModel[128];         /* station model */
unsigned long                 ulStationStates;      /* SI_STATIONSTATE_xxx flags */
unsigned long                 ulFunctions1;         /* Enum_SiFunction1xxx API functions supported */
unsigned long                 ulFunctions2;         /* Enum_SiFunction2xxx API functions supported */
TSSiCapabilitiesDisplay       DisplayCapabilities;  /* text display capabilities */
TSSiCapabilitiesRinger        RingerCapabilities;   /* ringer capabilities */
unsigned long                 ulAlerts;             /* supported alerts (Enum_SiAlertTypeXXX ) */
int                           iNumKeys;             /* number of supported keys */
int                           iNumIndicators;       /* number of supported indicators */
int                           iNumVolumeDevices;    /* number of devices with volume control */
int                           iNumSensitivityDevices;/* num sensitivity control devices */
int                           iNumAudioDevices;     /* number of media devices */
int                           iNumSoftKeys;         /* number of soft-keys */
unsigned long                 ulParameters;         /* supported parameters Enum_SiParmXXX flags */
} TSSiCapabilitiesMain;
```

```
typedef enum
{
        Enum_SiFunction1Invalid         = 0,
        Enum_SiFunction1None            = 0x80000000,   /* may not be combined with other TEnumSiFunction1 values */

        Enum_SiFunction1Open            = 0x00000001,
        Enum_SiFunction1Close           = 0x00000002,
        Enum_SiFunction1SetRinger       = 0x00000004,
        Enum_SiFunction1GetRinger       = 0x00000008,
        Enum_SiFunction1SendAlert       = 0x00000010,
        Enum_SiFunction1SetVolume       = 0x00000020,
        Enum_SiFunction1GetVolume       = 0x00000040,
        Enum_SiFunction1SetSensitivity  = 0x00000080,
        Enum_SiFunction1GetSensitivity  = 0x00000100,
        Enum_SiFunction1SetIndicator    = 0x00000200,
        Enum_SiFunction1GetIndicator    = 0x00000400,
        Enum_SiFunction1GetKeyState     = 0x00000800,
        Enum_SiFunction1SetLocalAudioRoute  = 0x00001000,
        Enum_SiFunction1GetLocalAudioRoute  = 0x00002000,
        Enum_SiFunction1DisplayClear    = 0x00004000,
        Enum_SiFunction1SetDisplayText  = 0x00008000,
        Enum_SiFunction1GetDisplayText  = 0x00010000,
        Enum_SiFunction1SetParm         = 0x00020000,
        Enum_SiFunction1GetParm         = 0x00040000,
        Enum_SiFunction1GetStationState = 0x00080000,
        Enum_SiFunction1SetStationState = 0x00100000,
        Enum_SiFunction1SetCallerId     = 0x00200000,
        Enum_SiFunction1SetCallTimer    = 0x00400000,
        Enum_SiFunction1SetSoftKeyText  = 0x00800000,
        Enum_SiFunction1GetSoftKeyText  = 0x01000000,
} TEnumSiFunction1;
```

```
typedef enum
{
        Enum_SiFunction2IUnvalid        = 0,
        Enum_SiFunction2None            = 0x80000000,   /* may not be combined with other TEnumSiFunction2 values */
} TEnumSiFunction2;
```

3.2.3 Example

#include “silib.h”

```
void main()
{
TSSiCapabilitiesMain StationCapabilities;
TSSiCapabilitiesKey    *pKeyCaps = NULL;
TSSiCapabilitiesIndicator     *pIndCaps = NULL;

        /* retrieve key and inidicator capabilities for the station */
        if (0 == si_GetStationCapabilities(
                                        “ssiB1C1”,
                                        Enum_SiCapabilitiesMain,
                                        &StationCapabilities, sizeof(TSSiCapabilitiesMain)))
        {
                pKeyCaps = (TSSiCapabilitiesKey*)malloc(StationCapabilities.nNumKeys*
                                                        sizeof(TSSiCapabilitiesKey));
                pIndCaps = (TSSiCapabilitiesIndicator*)malloc(StationCapabilities.nNumIndicators*
                                                        sizeof(TSSiCapabilitiesIndicator));
                si_GetStationCaps(     “ssiB1C1”,
                                        Enum_SiCapabilitiesKeys,
                                        pKeyCaps,
                                        StationCapabilities.nNumKeys*sizeof(TSSiCapabilitiesKey));
                si_GetStationCaps(     “ssiB1C1”,
                                        Enum_SiCapabilitiesIndicators,
                                        pIndCaps,
                                        StationCapabilities.nNumIndicators*
                                                                sizeof(TSSiCapabilitiesIndicator));
                /* print the capabilities … */

                free(pKeyCaps);
                free(pIndCaps);
        }

        return;
}
```

3.3 Ringer Control

The ringer control functions provide the application with the ability to enable and disable the ringer on the station. Control parameters include various ringer on/off cadence patterns supported by the station as well as various ring tone types that may be supported by the station. An application may enable the ringer on a station to provide inbound call alert, call-waiting alert, time alarm features, etc. The application uses the capabilities structure retrieval to determine the ringer cadences and tones supported by the specified station.

3.3.1 Functions

```
int si_SetRinger(int iHandle, const TSSiRingerState* pRingState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
const TSSiRingerState*	pRingState	- new ring state
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Changes the state of the ringer on the station. Controls ring cadence and ring tone. A cadence of 0 disables the station's ringer. Note that on some interfaces, not all cadences support all of the reported tones.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetRingerSuccess
Enum_SiEvSetRingerFail

Termination Success Event Data Pointer Contents

NULL

int si_GetRinger(int iHandle, TSSiRingerState* pRingState, TEnumSyncMode eSyncMode)

Inputs

int	iHandle	- station handle
TSSiRingerState*	piRingState	- ring state
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the current state of the station's ringer.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.

Termination Event

Enum_SiEvGetRingerSuccess
Enum_SiEvGetRingerFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiRingerState`** structure.

3.3.2 Types

```
typedef struct SSiCapabilitiesRinger
{
int                          iNumCadences;  /* num ring cadences (including 0-off) */
int                          iNumTones;     /* num ring tones */
} TSSiCapabilitiesRinger;

typedef struct SSiRingerState
{
int                          iCadence;      /* ring cadence to set (0-off) */
int                          iTone;         /* ring tone */
} TSSiRingerState;
```

3.4 Alert Control

The Alert control functions provide the application with the ability to enable Alert devices that may be supported by the specified stations. Some stations have the ability to generate local beep tones and local call-waiting tones. These functions provide the ability to play Alert tones that are generated locally by the specified station. The application uses the capabilities structure retrieval to determine the Alert tones that are supported by the specified station.

3.4.1 Functions

```
int si_SendAlert(int iHandle, TEnumSiAlertType eAlert, TEnumSyncMode
eSyncMode)
```

Inputs

int	iHandle	- station handle
TEnumSiAlertType	eAlert	- Enum_SiAlertTypeXXX
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Sends alert to the station. Some digital stations support the local rendering of beep tones, call-waiting tones, and other alerting tones. This function commands the station to generate the specified locally-rendered alert tone.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Invalid alert type specified.

Termination Event

Enum_SiEvSendAlertSuccess
Enum_SiEvSendAlertFail

Termination Success Event Data Pointer Contents

NULL

3.4.2 Types

```
typedef enum
{
      Enum_SiAlertTypeInvalid       = 0,
      Enum_SiAlertTypeNone          = 0x80000000,      /* may not be combined with other TEnumSiAlertType values */

      Enum_SiAlertTypeBeep          = 0x00000001,
      Enum_SiAlertTypeCallWaiting   = 0x00000002,

      Enum_SiAlertTypeOther1        = 0x00100000,
      Enum_SiAlertTypeOther2        = 0x00200000,
      Enum_SiAlertTypeOther3        = 0x00400000,
} TEnumSiAlertType;
```

3.5 Volume Control

The volume control functions provide the application with the ability to control the local audio volume of the station's various audio output devices (handset, speaker-phone). Some stations support the external control of the local audio volume, while other stations control the volume locally. The application uses the capabilities structure retrieval to determine the amount of external volume control supported by the specified station.

3.5.1 Functions

```
int si_SetVolume(int iHandle, TSSiVolumeDeviceState* pVolumeDeviceState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiVolumeDeviceState*	pVolumeDeviceState	- ptr to struct specifying device and level to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Sets the specified audio output device's local volume. Note: On some stations, volume control is completely local and not settable externally. The eDevice and iVolume fields of TSSiVolumeDeviceState must be set prior to calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetVolumeSuccess
Enum_SiEvSetVolumeFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetVolume(int iHandle, TSSiVolumeDeviceState* pVolumeDeviceState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiVolumeDeviceState*	pVolumeDeviceState	- ptr to struct specifying device state to retrieve
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the volume of the specified audio output device on the station. The eDevice field of TSSiVolumeDeviceState must be set prior to calling this function. The iVolume field of TSSiVolumeDeviceState will be filled upon completion of the function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetVolumeSuccess
Enum_SiEvGetVolumeFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiVolumeDeviceState`** structure.

Synchronous Example

```
#include <stdio.h>
#include <srllib.h>
#include <silib.h>

void main()
{
   int iHandle;

   /* Main Processing */

   /* ASSUMPTION: A valid iHandle was obtained from prior call to si_Open() of a channel device. */

   TSSiVolumeDeviceState VolumeState;
   VolumeState.eDevice = Enum_SiVolumeDeviceRinger;
   if (si_GetVolume(iHandle, &VolumeState, Enum_SyncMode) == SI_FAILURE)
   {
      printf("si_GetVolume failed - %s error = %d\n",
         ATDV_NAMEP(iHandle), ATDV_LASTERR(iHandle));

      /* Perform Error Processing */
   }
   else
   {
      printf("si_GetVolume(%s) device %d = %d\n",
         ATDV_NAMEP(iHandle), VolumeState.eDevice, VolumeState.iVolume);
   }
}
```

Asynchronous Example

```
#include <stdio.h>
#include <srllib.h>
#include <silib.h>

void main()
{
   int iHandle;

   /* Main Processing */

   /* ASSUMPTION: A valid iHandle was obtained from prior call to si_Open() of a channel device. */

   TSSiVolumeDeviceState VolumeState;
   VolumeState.eDevice = Enum_SiVolumeDeviceRinger;
   if (si_GetVolume(iHandle, &VolumeState, Enum_AsyncMode) == SI_FAILURE)
   {
      printf("si_GetVolume failed - %s error = %d\n",
         ATDV_NAMEP(iHandle), ATDV_LASTERR(iHandle));

      /* Perform Error Processing */
   }
}

void CheckEvent()
{
   int iEventType = sr_getevttype();
   int iDeviceID = sr_getevtdev();
   void* pvData = sr_getevtdatap();

   switch(iEventType)
   {
   /* Other events */

   /* Expected reply to si_SetVolume() */
   case Enum_SiEvGetVolumeSuccess:
      TSSiVolumeDeviceState *pVolumeState;

      pVolumeState = (TSSiVolumeDeviceState *) pvData;
      printf("Received Enum_SiEvGetVolumeSuccess for device = %s\n",
         ATDV_NAMEP(iDeviceID));
      printf("Volume device %d = %d\n", pVolumeState->eDevice, pVolumeState->iVolume);
      break;

   default:
      printf("Received unknown event = %d for device = %s\n",
         iEventType, ATDV_NAMEP(iDeviceID));
      break;
   }
}
```

3.5.2 Types

```
typedef enum
{
    Enum_SiVolumeDeviceInvalid              = 0,
    Enum_SiVolumeDeviceNone                 = 0x80000000,   /* may not be combined with other TEnumSiVolumeDevice values */

    Enum_SiVolumeDeviceRinger               = 0x00000001,
    Enum_SiVolumeDeviceHandsetSpeaker       = 0x00000002,
    Enum_SiVolumeDeviceIntercomSpeaker      = 0x00000004,
    Enum_SiVolumeDeviceSpeakerphoneSpeaker  = 0x00000008,
    Enum_SiVolumeDeviceHeadsetSpeaker       = 0x00000010,
} TEnumSiVolumeDevice;


typedef struct SSiCapabilitiesVolumeDevice
{
TEnumSiVolumeDevice   eDevice;              /* Enum_SiVolumeDeviceXXX */
int                   iMinVolume;           /* min device volume */
int                   iMaxVolume;           /* max device volume */
} TSSiCapabilitiesVolumeDevice;

typedef struct SSiVolumeDeviceState
{
TEnumSiVolumeDevice   eDevice;       /* specifies the volume device */
int                   iVolume;       /* specifies the volume level of the device */
} TSSiVolumeDeviceState;
```

3.6 Sensitivity Control

The sensitivity control functions provide the application with the ability to control the local audio sensitivity of the station's various audio input devices (handset microphone, speaker-phone microphone). Some stations support the external control of the local audio sensitivity, whereas other stations control the sensitivity locally. The application uses the capabilities structure retrieval to determine the amount of external sensitivity control supported by the specified station.

3.6.1 Functions

```
int si_SetSensitivity(int iHandle, TSSiSensitivityDeviceState*
pSensitivityDeviceState, TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiSensitivityDeviceState*	pSensitivityDeviceState	- ptr to struct specifying device state to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Sets the specified audio input device's local sensitivity. Note: On some stations, sensitivity control is completely local and not capable of being set externally. The eDevice and iSensitivity fields of TSSiSensitivityDeviceState must be set prior to calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetSensitivitySuccess
Enum_SiEvSetSensitivityFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetSensitivity(int iHandle, TSSiSensitivityDeviceState*
pSensitivityDeviceState, TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiSensitivityDeviceState*	pSensitivityDeviceState	- ptr to struct specifying device state to retrieve
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the sensitivity of the specified audio input device on the station. Parameter device of sensitivity_state is set on input by application. The eDevice field of TSSiSensitivityDeviceState must be set prior to calling this function. The iSensitivity field of TSSiSensitivityDeviceState will be filled upon completion of the function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetSensitivitySuccess
Enum_SiEvGetSensitivityFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiSensitivityDeviceState`** structure.

3.6.2 Types

```
typedef enum
{
    Enum_SiSensitivityDeviceInvalid          = 0,
    Enum_SiSensitivityDeviceNone             = 0x80000000,   /* may not be combined with other TEnumSiSensitivityDevice values */

    Enum_SiSensitivityDeviceHandsetMic       = 0x00000001,
    Enum_SiSensitivityDeviceSpeakerphoneMic  = 0x00000002,
    Enum_SiSensitivityDeviceHeadsetMic       = 0x00000004,
} TEnumSiSensitivityDevice;


typedef struct SSiCapabilitiesSensitivityDevice
{
TEnumSiSensitivityDevice      eDevice;              /* Enum_SiSensitivityDeviceXXX
int                           iMinSensitivity;      /* min device sensitivity */
int                           iMaxSensitivity;      /* max device sensitivity */
} TSSiCapabilitiesSensitivityDevice;

typedef struct SSiSensitivityDeviceState
{
TEnumSiSensitivityDevice      eDevice;       /* Enum_SiSensitivityDeviceXXX
int                           iSensitivity;  /* device sensitivity */
} TSSiSensitivityDeviceState;
```

3.7 Indicator Control

The indicator control functions provide the application with the ability to control the indicators on the station. An indicator may be a lamp, a LED, or an icon. Different stations support different numbers and types of indicators. Some indicators are comprised of multiple sub-indicators, each with a different LED color. Some indicators are comprised of a single indicator that can be enabled in different colors. Indicators may be used to signal inbound calls, held calls, active calls, message-waiting, external party activity, etc. The application uses the capabilities structure retrieval to determine the number and types of indicators supported by the specified station. An application can determine if an indicator and a key are co-located if the indicator and key have the same identifier number (SI_INDICATORID_xxx, SI_KEYID_xxx).

3.7.1 Functions

```
int si_SetIndicator(int iHandle, const TSSiIndicatorState*
pIndicatorState, TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
const TSSiIndicatorState*	pIndicatorState	- indicator state to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Sets the state of the specified indicator on the station. The ulIndicatorId and eSubIndicator fields of the TSSiIndicatorState object must be set by the application before calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorInvIndicatorId	Invalid indicator is specified.
Enum_SiErrorInvSubIndicator	Invalid sub-indicator is specified.
Enum_SiErrorInvColor	Invalid indicator color is specified.
Enum_SiErrorInvState	Invalid indicator state is specified.

Termination Event

Enum_SiEvSetIndicatorSuccess
Enum_SiEvSetIndicatorFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetIndicator(int iHandle, TSSiIndicatorState* pIndicatorState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiIndicatorState*	pIndicatorState	- indicator state to get
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the current state of the specified lamp on the station. The ulIndicatorId and eSubIndicator fields of the TSSiIndicatorState object must be set by the application before calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorInvIndicatorId	Invalid indicator is specified.
Enum_SiErrorInvSubIndicator	Invalid sub-indicator is specified.

Termination Event

Enum_SiEvGetIndicatorSuccess
Enum_SiEvGetIndicatorFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiIndicatorState`** structure.

3.7.2 Types

```
#define SI_MAXNUM_SUBINDICATORS      4
typedef struct SSiCapabilitiesIndicator
{
unsigned long                ulIndicatorId;        /* Enum_SiIndicatorIdXXX */
int                          iNumSubIndicators;    /* number of sub-indicators */
TSSiCapabiltiesSubIndicator  capSubIndicator[SI_MAXNUM_SUBINDICATORS];    /* indicator device capabilities */
} TSSiCapabilitiesIndicator;

Note: The ulIndicatorId field of TSSiCapabilitiesIndicator and TSSiIndicatorState is an unsigned long instead of
a TEnumSiIndicatorId because applications need to add to Enum_SiIndicatorLine1/Soft1 to reach other line and
soft indicators. For instance, to set the indicator for line 10:
       ulLine10 = (unsigned long)Enum_SiIndicatorLine1+9

typedef struct SSiCapabiltiesSubIndicator
{
TEnumSiSubIndicator          eSubIndicator;        /* Enum_SiSubIndicatorXXX */
unsigned long                ulStates;             /* Enum_SiIndicatorStateXXX flags */
unsigned long                ulColors;             /* Enum_SiIndicatorColorXXX flags */
} TSSiCapabiltiesSubIndicator

typedef struct SSiIndicatorState
{
unsigned long                ulIndicatorId;        /* Enum_SiIndicatorIdXXX */
TEnumSiSubIndicator          eSubIndicator;        /* Enum_SiSubIndicatorXXX */
TEnumSiIndicatorState        eState;               /* Enum_SiIndicatorStateXXX */
TEnumSiIndicatorColor        eColor;               /* Enum_SiIndicatorColorXXX */
} TSSiIndicatorState;
```

Sub-Indicators

```
typedef enum
{
    Enum_SiSubIndicatorInvalid       = 0,
    Enum_SiSubIndicatorNone          = 0x80000000,  /* may not be combined with other TEnumSiSubIndicator values */

    Enum_SiSubIndicatorIndicator     = 0x00000001,  /* indicator (may be multiples on each indicator) */
    Enum_SiSubIndicatorSelector      = 0x00000002,  /* selector part (used as an indicator selector) */
} TEnumSiSubIndicator;
```

Indicator IDs

```
typedef enum
{
       Enum_SiIndicatorIdInvalid            = 0,
       Enum_SiIndicatorIdNone               = 0x80000000,  /* may not be combined with other TEnumSiIndicatorId values */

       Enum_SiIndicatorIdHookswitch         = (0x0011),    /* hook-switch */
        Enum_SiIndicatorIdHold              = (0x0012),    /* hold */
        Enum_SiIndicatorIdConference        = (0x0013),    /* conference */
        Enum_SiIndicatorIdTransfer          = (0x0014),    /* transfer */

        Enum_SiIndicatorIdLine1             = (0x0015),    /* line indicators */
        Enum_SiIndicatorIdLine999           = (0x03FB),
        Enum_SiIndicatorIdFunc1             = (0x03FC),    /* assignable function/feature keys */
        Enum_SiIndicatorIdFunc999           = (0x07E2),


        Enum_SiIndicatorIdSpeaker           = (0x0800),    /* speaker */
        Enum_SiIndicatorIdMsg               = (0x0801),    /* message */
        Enum_SiIndicatorIdItcm              = (0x0802),    /* intercom */
        Enum_SiIndicatorIdRelease           = (0x0803),    /* drop/release */

        Enum_SiIndicatorIdSoft1             = (0x0900),    /* soft-keys */
```

```
        Enum_SiIndicatorIdSoft999           = (0x0CE6),

        Enum_SiIndicatorIdMute              = (0x0D00),    /* mute */

        Enum_SiIndicatorIdOther1            = (0x00100000),       /* extensions */
        Enum_SiIndicatorIdOther2            = (0x00200000),       /* extensions */
        Enum_SiIndicatorIdOther3            = (0x00400000),       /* extensions */
} TEnumSiIndicatorId;
```

Indicator Colors

```
typdef enum
{
        Enum_SiIndicatorColorInvalid        0,
        Enum_SiIndicatorColorNone           0x80000000,     /* may not be combined with other TEnumSiIndicatorColor values */

        Enum_SiIndicatorColorRed            0x00000001,
        Enum_SiIndicatorColorGreen          0x00000002,
        Enum_SiIndicatorColorYellow         0x00000004,
        Enum_SiIndicatorColorOrange         0x00000008,
        Enum_SiIndicatorColorBlue           0x00000010,
        Enum_SiIndicatorColorBlack          0x00000020,

        Enum_SiIndicatorColorOther1         0x00100000,
        Enum_SiIndicatorColorOther2         0x00200000,
        Enum_SiIndicatorColorOther3         0x00400000,
} TEnumSiIndicatorColor;
```

Indicator States

```
typdef enum
{
        Enum_SiIndicatorStateInvalid        0,
        Enum_SiIndicatorStateNone           0x80000000,     /* may not be combined with other TEnumSiIndicatorState values */

        Enum_SiIndicatorStateOff            0x00000001,
        Enum_SiIndicatorStateSteady         0x00000002,
        Enum_SiIndicatorStateBlink          0x00000004,
        Enum_SiIndicatorStateFastBlink      0x00000008,
        Enum_SiIndicatorStateSlowBlink      0x00000010,

        Enum_SiIndicatorStateOther1         0x00100000,
        Enum_SiIndicatorStateOther2         0x00200000,
        Enum_SiIndicatorStateOther3         0x00400000,
} TEnumSiIndicatorState;
```

3.8 Key Control

The key control functions provide the application with the ability to monitor key presses and releases made by the user of the station. A key press may be used to dial numbers, activate a call-appearance, initiate a function such as transfer or conference, etc. Some keys on stations generate both a down event and an up event. Some keys on stations only provide a down or an up event. The application uses the capabilities structure retrieval to determine the number and types of keys supported by the specified station and the capabilities of each key. An application can determine if an indicator and a key are co-located if the indicator and key have the same identifier number (Enum_SiIndicatorIdXXX, Enum_SiKeyIdXXX).

3.8.1 Functions

```
int si_GetKeyState(int iHandle, TSSiKeyState* pKeyState, TEnumSyncMode
eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiKeyState*	pKeyState	- keystate to get
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the state of the specified key. Some keys generate a down and up state, such as dial keys. This function returns the current state of the key, whether it is up or down. On input, ulKeyId of TSSiKeyState object must be set. On successful completion, the remainder of TSSiKeyState is filled.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorInvKeyId	Invalid key is specified.

Termination Event

Enum_SiEvGetKeyStateSuccess
Enum_SiEvGetKeyStateFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiKeyState`** structure.

Unsolicited Event

Enum_SiEvKeyState - key state change, event data – TSSiKeyState

Unsolicited Event Data Pointer Contents

Pointer to **TSSiKeyState** structure.

```
int si_GetSoftKeyText(int iHandle, TSSiSoftKeyText* pSoftKeyText,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiSoftKeyText*	pSoftKeyText	- soft key text parameters
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Retrieves text currently associated with the specified soft-key. The ulKeyId field, and iLen fields of TSSiSoftKeyText must be set prior to calling this function. The iLen field must be set to the maximum number of bytes to retrieve. If called in synchronous mode, the pszText field must be set to an application buffer prior to calling this function. If called in asynchronous mode, the pszText field is ignored.

Upon completion of this function, the pszText buffer will be filled with the contents of the soft-key text, and the iLen field will contain the length of the data copied into the pszText buffer.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorInvLen	Invalid length is specified.
Enum_SiErrorInvKeyId	Invalid soft-key is specified.

Termination Event

Enum_SiEvGetSoftKeyTextSuccess
Enum_SiEvGetSoftKeyTextFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiSoftKeyText`** structure.

```
int si_SetSoftKeyText(int iHandle, TSSiSoftKeyText* pSoftKeyText,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiSoftKeyText*	pSoftKeyText	- soft key text parameters
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Sets text currently associated with the specified soft-key. The ulKeyId field, pszText field, and iLen fields of TSSiSoftKeyText must be set prior to calling this function. The iLen field must be set to the number of bytes of valid data pointed to by the pszText field.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorInvLen	Invalid length is specified.
Enum_SiErrorInvKeyId	Invalid soft-key is specified.

Termination Event

Enum_SiEvSetSoftKeyTextSuccess
Enum_SiEvSetSoftKeyTextFail

Termination Success Event Data Pointer Contents

NULL

3.8.2 Types

```
typedef struct SSiCapabilitiesKey
{
unsigned long          ulKeyId;       /* Enum_SiKeyIdXXX */
unsigned long          ulStates;      /* Enum_SiKeyStateXXX flags */
} TSSiCapabilitiesKey

typedef struct SSiKeyState
{
unsigned long          ulKeyId;       /* Enum_SiKeyIdXXX */
TEnumSiKeyState        eState;        /* Enum_SiKeyState*/
int                    iMsDuration;   /* ms duration of down (only on SI_KEYSTATE_UP) */
} TSSiKeyState
```

Note: The ulKeyId field of TSSiCapabilitiesKey and TSSiKeyState is an unsigned long instead of a TEnumSiKeyId because applications need to add to Enum_SiKeyLine1/Soft1 to reach other line and soft keys. For instance, to get the key state for line 10:

```
        ulLine10 = (unsigned long)Enum_SiKeyLine1+9

typedef struct SSiCapabilitiesSoftKey
{
unsigned long          ulMaxTextLen;  /* maximum text length */
} TSSiCapabilitiesSoftKey

typedef struct SSiSoftKeyText
{
unsigned long          ulKeyId;       /* Enum_SiKeyIdXXX */
int                    iLen;          /* length of pszText buffer */
char*                  pszText;       /* pointer to application buffer of soft-key text */
} TSSiSoftKeyText;
```

Key IDs

```
typedef enum
{
        Enum_SiKeyIdInvalid           = 0,
        Enum_SiKeyIdNone              = 0x80000000,   /* may not be combined with other TEnumSiKeyId values */


        Enum_SiKeyId0                 = (0x0001),     /* key-pad '0' */
        Enum_SiKeyId1                 = (0x0002),     /* key-pad '1' */
        Enum_SiKeyId2                 = (0x0003),     /* key-pad '2' */
        Enum_SiKeyId3                 = (0x0004),     /* key-pad '3' */
        Enum_SiKeyId4                 = (0x0005),     /* key-pad '4' */
        Enum_SiKeyId5                 = (0x0006),     /* key-pad '5' */
        Enum_SiKeyId6                 = (0x0007),     /* key-pad '6' */
        Enum_SiKeyId7                 = (0x0008),     /* key-pad '7' */
        Enum_SiKeyId8                 = (0x0009),     /* key-pad '8' */
        Enum_SiKeyId9                 = (0x000A),     /* key-pad '9' */
        Enum_SiKeyIdStar              = (0x000B),     /* key-pad '*' */
        Enum_SiKeyIdPound             = (0x000C),     /* key-pad '#' */
        Enum_SiKeyIdA                 = (0x000D),     /* key-pad 'A' or 'a' */
        Enum_SiKeyIdB                 = (0x000E),     /* key-pad 'B' or 'b' */
        Enum_SiKeyIdC                 = (0x000F),     /* key-pad 'C' or 'c' */
        Enum_SiKeyIdD                 = (0x0010),     /* key-pad 'D' or 'd' */

        Enum_SiKeyIdHookswitch        = (0x0011),     /* hook-switch */
        Enum_SiKeyIdHold              = (0x0012),     /* hold*/
        Enum_SiKeyIdConference        = (0x0013),     /* conference */
        Enum_SiKeyIdTransfer          = (0x0014),     /* transfer */

        Enum_SiKeyIdLine1             = (0x0015),     /* line keys */
        Enum_SiKeyIdLine999           = (0x03FB),
        Enum_SiKeyIdFunc1             = (0x03FC),     /* assignable function/feature keys */
        Enum_SiKeyIdFunc999           = (0x07E2),
```

```
        Enum_SiKeyIdSpeaker          = (0x0800),    /* speaker */
        Enum_SiKeyIdMsg              = (0x0801),    /* message*/
        Enum_SiKeyIdItcm             = (0x0802),    /* intercom*/
        Enum_SiKeyIdRelease          = (0x0803),    /* drop/release */

        Enum_SiKeyIdSoft1            = (0x0900),    /* soft-keys */
        Enum_SiKeyIdSoft999          = (0x0CE6),    /* soft-keys */

        Enum_SiKeyIdMute             = (0x0D00),    /* mute */

        Enum_SiKeyIdOther1           = (0x0010000), /* extensions*/
        Enum_SiKeyIdOther2           = (0x0020000), /* extensions*/
        Enum_SiKeyIdOther3           = (0x0040000), /* extensions*/
} TEnumSiKeyId;
```

Key States

```
typedef enum
{
       Enum_SiKeyStateInvalid        = 0,
       Enum_SiKeyStateNone           = 0x80000000,   /* may not be combined with other TEnumSiKeyState values */

       Enum_SiKeyStateUp             = 0x00000001,
       Enum_SiKeyStateDown           = 0x00000002,
} TEnumSiKeyState;
```

3.9 Local Audio Routing Control

The local audio routing control functions provide the application with the ability to enable/disable local audio routes on the station. For instance, some stations must have their speaker-phone speaker externally enabled in order for the user to hear the audio being played to the station. These functions give the application the ability to control the selection of the local audio input and output devices on the station. The application uses the capabilities structure retrieval to determine the number and types of audio devices supported by the specified station and the capabilities of each device.

3.9.1 Functions

```
int si_SetLocalAudioRoute(int iHandle, TSSiAudioRoute* pAudioRoute,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiAudioRoute*	pAudioRoute	- audio route mode to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Enables/disables audio input/output devices on the station. New audio mode set by the application replaces previous. The eDevice and eMode fields of TSSiAudioRoute structure must be set prior to calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetLocalAudioRouteSuccess
Enum_SiEvSetLocalAudioRouteFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetLocalAudioRoute(int iHandle, TSSiAudioRoute* pAudioRoute,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiAudioRoute*	pAudioRoute	- audio route mode to get
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the current enabled/disabled state of the specified audio input/output device on the station. The eDevice field of TSSiAudioRoute structure must be set prior to calling this function. The eMode field of TSSiAudioRoute will be filled upon completion of the function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetLocalAudioRouteSuccess
Enum_SiEvGetLocalAudioRouteFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiAudioRoute`** structure.

3.9.2 Types

```
typedef enum
{
    Enum_SiAudioModeInvalid          = (0),
    Enum_SiAudioModeNone             = (0x80000000),        /* may not be combined with other TEnumSiAudioMode values */

    Enum_SiAudioModeBidirection      = (0x0001),
    Enum_SiAudioModeMuteFromStation  = (0x0002),
} TEnumSiAudioMode;

typedef enum
{
    Enum_SiAudioDeviceInvalid        = (0),
    Enum_SiAudioDeviceNone           = (0x80000000),        /* may not be combined with other TEnumSiAudioDevice values */

    Enum_SiAudioDeviceHandset        = (0x0001),
    Enum_SiAudioDeviceSpeakerphone   = (0x0002),
    Enum_SiAudioDeviceHeadset        = (0x0004),
} TEnumSiAudioDevice;


typedef struct SSiCapabilitiesAudioDevice
{
TEnumSiAudioDevice    eDevice;             /* Enum_SiAudioDeviceXXX device type */
unsigned long         ulModes;             /* Enum_SiAudioModeXXX  flags that can be controlled */
} TSSiCapabilitiesAudioDevice;

typedef struct SSiAudioRoute
{
TEnumSiAudioDevice    eDevice;             /* Enum_SiAudioDeviceXXX device type */
TEnumSiAudioMode      eMode;               /* Enum_SiAudioModeXXX audio route mode */
} TSSiAudioRoute;
```

3.10 Text Display Control

The text display control functions provide the application with the ability to set the contents of the station's text display. These functions give the application full control over the contents and layout of the station's text display. Unicode is used as the format of the text data in order to support stations that supported extended character sets. The text display may be used by the application to display call party information, instant text messages, call queue statistics, etc. The application uses the capabilities structure retrieval to determine the capabilities of the station's time display.

3.10.1 Functions

```
int si_DisplayClear(int iHandle, int iPage, int iRow, TEnumSyncMode
eSyncMode)
```

Inputs:

int	iHandle	- station handle
int	iPage	- display page (1-based, 0 for all)
int	iRow	- row to clear (1-based, 0 for all)
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

If iPage is 0, iRow is ignored and all display contents are cleared and sets page/row/col cursor position to 1,1,1. If iPage is >= 0, clears the specified display page, but does not change current cursor position. If iRow is >= 0, clears the specified row on the specified page, but does not change current cursor position.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.

Termination Event

Enum_SiEvDisplayClearSuccess
Enum_SiEvDisplayClearFail

Termination Success Event Data Pointer Contents

NULL

```
int si_SetDisplayActivePage(int iHandle, int iPage, TEnumSyncMode
eSyncMode)
```

Inputs:

int	iHandle	- station handle
int	iPage	- display page (1-based)
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Sets the active display page.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Page is invalid.

Termination Event

Enum_SiEvSetDisplayActivePageSuccess
Enum_SiEvSetDisplayActivePageFail

Termination Success Event Data Pointer Contents

NULL

```
int si_SetDisplayText(int iHandle, TSSiDisplayText* pDisplayText,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiDisplayText*	pDisplayText	- display text to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Sets the state of the station's text display. The iPage, iRow, iCol, iLen, ulAttribute, and pszText fields of TSSiDisplayText must be set prior to calling this function. The iLen field must be set to the length of the application buffer pointed to by pszText.

If the iPage field is 0, then the current active page is updated.

If the iRow field is 0, then current row cursor position is used. If the iCol field is 0, then current column cursor position is used.

If the pszText field is NULL or the iLen field is 0, then the cursor position is moved to the specified iRow and iCol (if > 0).

Text wrapping not provided. Truncation occurs at end of row.

ulAttribute applies to all characters in current text display string. Multiple Enum_SiTextDisplayAttrib flags may be OR'd together in the ulAttribute argument to apply multiple attributes to the same text (as supported by the underlying device).

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetDisplayTextSuccess
Enum_SiEvSetDisplayTextFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetDisplayText(int iHandle, TSSiDisplayText* pDisplayText,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiDisplayText*	pDisplayText	- display text to get
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Retrieves the current state of the station's text display. The iPage, iRow, iCol, and iLen fields of TSSiDisplayText must be set prior to calling this function. The iLen field must be set to the maximum number of bytes to retrieve (including the NULL terminator). If called in synchronous mode, the pszText field must be set to an application buffer prior to calling this function. If called in asynchronous mode, the pszText field is ignored.

Upon completion of the function, the pszText buffer is filled with the contents of the requested display section. The iLen field will contain the number of bytes of data copied into the pszText buffer plus the NULL terminator.

To retrieve entire display, set the iRow field to 0, the iCol field to to 0, and a pszText buffer big enough to contain entire display (determined by capabilities).

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorInvLen	Invalid length is specified.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetDisplayTextSuccess
Enum_SiEvGetDisplayTextFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiDisplayText`** structure.

Synchronous Example

```
#include <stdio.h>
#include <srllib.h>
#include <silib.h>

void main()
{
   int iHandle;

   /* Main Processing */

   /* ASSUMPTION: A valid iHandle was obtained from prior call to si_Open() of a channel device. */

   TSSiDisplayText DisplayText;
   char szDisplayText[128];
   DisplayText.iPage = 0;               /* current active page */
   DisplayText.iRow = 0;                /* entire display */
   DisplayText.iCol = 0;                /* entire display */
   DisplayText.iLen = 128;              /* maximum bytes to retrieve */
   DisplayText.pszText = szDisplayText; /* retrieve buffer */

   if (si_GetDisplayText(iHandle, &DisplayText, Enum_SyncMode) == SI_FAILURE)
   {
      printf("si_GetDisplay failed - %s error = %d\n",
         ATDV_NAMEP(iHandle), ATDV_LASTERR(iHandle));

      /* Perform Error Processing */
   }
   else
   {
      printf("si_GetDisplay(%s) |%s|\n",
         ATDV_NAMEP(iHandle), DisplayText.pszText);
   }
}
```

Asynchronous Example

```
#include <stdio.h>
#include <srllib.h>
#include <silib.h>

void main()
{
   int iHandle;

   /* Main Processing */

   /* ASSUMPTION: A valid iHandle was obtained from prior call to si_Open() of a channel device. */

   TSSiDisplayText DisplayText;
   DisplayText.iPage = 0;               /* current active page */
   DisplayText.iRow = 0;                /* entire display */
   DisplayText.iCol = 0;                /* entire display */
   DisplayText.iLen = 128;              /* maximum bytes to retrieve */
   DisplayText.pszText = NULL;          /* buffer will be allocated by SI */
   if (si_GetDisplay(iHandle, &DisplayText, Enum_AsyncMode) == SI_FAILURE)
   {
      printf("si_GetDisplay failed - %s error = %d\n",
         ATDV_NAMEP(iHandle), ATDV_LASTERR(iHandle));

      /* Perform Error Processing */
   }

   /* other processing, wait for event completion */
}

void CheckEvent()
{
   int iEventType = sr_getevttype();
   int iDeviceID = sr_getevtdev();
   void* pvData = sr_getevtdatap();

   switch(iEventType)
   {
   /* Other events */

   /* Expected reply to si_GetDisplay() */
   case Enum_SiEvGetDisplaySuccess:
      TSSiDisplayText *pDisplayText;

      pDisplayText = (TSSiDisplayText *) pvData;
      printf("Received Enum_SiEvGetDisplaySuccess for device = %s\n",
         ATDV_NAMEP(iDeviceID));
      printf("Display len %d |%s|\n", pDisplayText->iLen, pDisplayText->pszText);
      break;

   default:
```

```
      printf("Received unknown event = %d for device = %s\n",
         iEventType, ATDV_NAMEP(iDeviceID));
      break;
   }
}
```

```
int si_GetDisplayCursorPos(int iHandle, TSSiDisplayCursorPosition*
pCursorPosition, TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiDisplayCursorPosition*	pCursorPosition	- cursor position retrieval parameters
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Retrieves the current cursor position of the specified page. The iPage field of TSSiDisplayCursorPosition must be set prior to calling this function. Upon completion of this function, the iCurrentRow and iCurrentCol fields of TSSiDisplayCursorPosition will be filled.

If iPage is specified as 0, the cursor position of the currently-active display page will be retrieved.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetDisplayCursorPosSuccess
Enum_SiEvGetDisplayCursorPosFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiDisplayCursorPosition`** structure.

```
int si_SetCallTimer(int iHandle, TSSiCallTimerState* pCallTimerState,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
TSSiCallTimerState*	pCallTimerState	- call timer state to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Enables the display of a call duration timer on the station's text display. This can be used by the application to enable call duration timing in the station's text display. While the call timer is enabled on a station, display updates can be sent to the station by the application, however, the call timer will be superimposed on any new display text.

When the call timer is stopped by the application, the station's text display is NOT automatically cleared.

The eAction field of TSSiCallTimerState must be set prior to calling this function. If eAction = Enum_SiCallTimerActionStart, then the eFormat and eLocation fields must also be set prior to calling this function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetCallTimerSuccess
Enum_SiEvSetCallTimerFail

Termination Success Event Data Pointer Contents

NULL

3.10.2 Types

```
typedef enum
{
        Enum_SiTextDisplayAttribInvalid      = 0,
        Enum_SiTextDisplayAttribNone         = 0x80000000,   /* may not be combined with other TEnumSiTextDisplayAttrib values */

        Enum_SiTextDisplayAttribBlink        = 0x00000001,
        Enum_SiTextDisplayAttribInvert       = 0x00000002,
        Enum_SiTextDisplayAttribUnderline    = 0x00000004,
} TEnumSiTextDisplayAttrib;

typedef enum
{
        Enum_SiCallTimerActionInvalid        = 0,
        Enum_SiCallTimerActionNone           = 0x80000000,   /* may not be combined with other TEnumSiCallTimerAction values */

        Enum_SiCallTimerActionStart          = 0x00000001,
        Enum_SiCallTimerActionStop           = 0x00000002,
        Enum_SiCallTimerActionRestart        = 0x00000004,
} TEnumSiCallTimerAction;

typedef enum
{
        Enum_SiCallTimerFormatInvalid        = 0,
        Enum_SiCallTimerFormatNone           = 0x80000000,   /* may not be combined with other TEnumSiCallTimerFormat values */

        Enum_SiCallTimerFormatDefault        = 0x00000001,   /* “HH:MM:SS” */
        Enum_SiCallTimerFormatHHMMSS         = 0x00000002,   /* “HH:MM:SS” – 24 hour, rollover at 24 */
        Enum_SiCallTimerFormatMSS            = 0x00000004,   /* “M:SS”      - minutes and seconds */
} TEnumSiCallTimerFormat;

typedef enum
{
        Enum_SiCallTimerLocationInvalid      = 0,
        Enum_SiCallTimerLocationNone         = 0x80000000,   /* may not be combined with other TEnumSiCallTimerLocation values */

        Enum_SiCallTimerLocationTopLeft      = 0x00000001,
        Enum_SiCallTimerLocationTopRight     = 0x00000002,
        Enum_SiCallTimerLocationBottomLeft   = 0x00000004,
        Enum_SiCallTimerLocationBottomRight  = 0x00000008,
} TEnumSiCallTimerLocation;
```

```
typedef struct SSiCapabilitiesTextDisplay
{
int                   iNumPages;                  /* num pages */
int                   iNumRows;                   /* num rows */
int                   iNumCols;                   /* num columns */
unsigned long         ulAttributes;               /* Enum_SiTextDisplayAttribXXX flags */

unsigned long         ulCallTimerActions;         /* call timer actions supported */
unsigned long         ulCallTimerFormats;         /* call timer formats supported */
unsigned long         ulCallTimerLocations;       /* call timer location in display */
} TSSiCapabilitiesTextDisplay;

typedef struct SSiCallTimerState
{
TEnumSiCallTimerAction       eAction;       /* action to execute on call timer */
TEnumSiCallTimerFormat       eFormat;       /* format (used if eAction = Enum_SiCallTimerActionStart) */
TEnumSiCallTimerLocation     eLocation;     /* location (used if eAction = Enum_SiCallTimerActionStart ) */
unsigned long                ulTimeSec;     /* specifies current call timer time, in seconds (0 – 86399) */
} TSSiCallTimerState;

typedef struct SSiDisplayText
{
int            iPage;               /* display page to set/get (1-based) */
int            iRow;                /* starting row of text to set/get (1-based) */
int            iCol;                /* starting col of text to set/get (1-based) */
unsigned long  ulAttribute;         /* text attribute */
int            iLen;                /* length of pszText buffer */
char*          pszText;             /* pointer to application buffer */
} TSSiDisplayText;


typedef struct SSiDisplayCursorPosition
{
int            iPage;               /* display page (1-based) */
int            iCurrentRow;         /* current row position of cursor on the page (1-based) */
int            iCurrentCol;         /* current col position of cursor on the page (1-based) */
} TSSiDisplayCursorPosition;
```

3.11 Caller-ID Control

The caller-id control functions provide the application with the ability to send caller-id information to stations that support FSK caller-id, DTMF TypeI and DTMF Type II caller-id. The application uses the capabilities structure retrieval to determine if the station supports FSK caller-id.

3.11.1 Functions

```
int si_SetCallerId(int iHandle, const TSSiCallerID* pCallerId,
TEnumSyncMode eSyncMode)
```

Inputs:

int	iHandle	- station handle
const TSSiCallerId*	pCallerId	- caller id information to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description:

Send caller-id information to the station. This function is applicable only to stations that use FSK to receive and display caller-id information.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorNoCT	Station device is not connected.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.

Termination Event

Enum_SiEvSetCallerIdSuccess
Enum_SiEvSetCallerIdFail

Termination Success Event Data Pointer Contents

NULL

3.11.2 Types

```
typedef struct SSiCallerId
{
char        month[]                 *  month
char        day[]                   * day of the month
char        hour[]                  * hour in local military time
char        minute[]                * minutes after the hour
char        name[]                  * calling party's directory name ("O", "P")
char        number[]                * calling party's directory number ("O", "P")
} TSSiCallerId;
```

3.12 Parameter Control

The parameter control functions provide the application with the ability to set run-time station operational parameters. The application uses the capabilities structure retrieval to determine the run-time operational parameters supported by the station

3.12.1 Functions

```
int si_SetParm(int iHandle, TSSiParameter* pParameter, TEnumSyncMode
eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiParameter*	pParameter	- pointer to parameter to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Sets specified parameter. The eParameter field, the iLen field, and the pvData field of TSSiParameter must be set prior to calling this function. The iLen field must be set to the length of the application buffer pointed to by pvData.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for a list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorInvLen	Invalid length is specified.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetParmSuccess
Enum_SiEvSetParmFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetParm(int iHandle, TSSiParameter* pParameter, TEnumSyncMode
eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiParameter*	pParameter	- pointer to parameter to retrieve
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves specified parameter. The eParameter field, the iLen field, and the pvData field of TSSiParameter must be set prior to calling this function. The iLen field must be set to the length of the application buffer pointed to by pvData. Upon completion of the function, the structure pointed to by pvData contains the value of the requested parameter.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetParmSuccess
Enum_SiEvGetParmFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiParameter`** structure.

3.12.2 Types

```
typedef enum
{
       Enum_SiParmInvalid            = 0,
       Enum_SiParmNone               = 0x80000000,   /* may not be combined with other TEnumSiParm values */

       Enum_SiParmDigitMode          = 0x00000001,
       Enum_SiParmUserContext        = 0x00000002,
} TEnumSiParm;

typedef enum
{
       Enum_SiDigitModeInvalid       = 0,
       Enum_SiDigitModeNone          = 0x80000000,   /* may not be combined with other TEnumSiDigitMode values */

       Enum_SiDigitModeDtmf          = 0x00000001,
       Enum_SiDigitModeOutOfBand     = 0x00000002,
} TEnumSiDigitMode;

typedef struct SSiParameter
{
TEnumSiParm    eParameter;    /* parameter to set/retrieve */
int            iLen;          /* length of buffer pointed to by pvData */
const void*    pvData;        /* pointer to application buffer containing parameter structure */
} TSSiParameter;

typedef struct SSiDigitMode
{
       TEnumSiDigitMode      eDigitMode;    /* Digit Mode */
} TSSiDigitMode;
```

3.13 Station State Control

The station state control functions provide the application with the ability to enable or disable a specified station. This may be used to busy-out a station based on server configuration. The application uses the capabilities structure retrieval to determine if the station can be externally enabled or disabled.

3.13.1 Functions

```
int si_SetStationState(int iHandle, TSSiStationState* pStationState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiStationState*	pStationState	- station state to set
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Enables/disables the specified station. The eState field of TSSiStationState must be set prior to calling this function. The eService field is ignored.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvSetStationStateSuccess
Enum_SiEvSetStationStateFail

Termination Success Event Data Pointer Contents

NULL

```
int si_GetStationState(int iHandle, TSSiStationState* pStationState,
TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TSSiStationState*	pStationState	- station state to retrieve
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves the current state of the specified station, whether it has been enabled, and whether a connection to the station has been established. Upon completion of this function, the eState and eService fields of TSSiStationState are filled.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorNotSupported	Device does not support this command.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFieldOutOfRange	Field in the structure is invalid.

Termination Event

Enum_SiEvGetStationStateSuccess
Enum_SiEvGetStationStateFail

Termination Success Event Data Pointer Contents

Pointer to **`TSSiStationState`** structure.

Unsolicited Events

Enum_SiEvDeviceServiceIn	- station is in-service – event data none
Enum_SiEvDeviceServiceOut	- station is out-of-service – event data none

Unsolicited Event Data Pointer Contents

NULL

Types

```
typedef enum
{
        Enum_SiStationStateInvalid          = 0,
        Enum_SiStationStateNone             = 0x80000000,   /* may not be combined with other TEnumSiStationState values */

        Enum_SiStationStateEnabled          = 0x00000001,
        Enum_SiStationStateDisabled         = 0x00000002,
} TEnumSiStationState;



typedef enum
{
        Enum_SiServiceStateInvalid          = 0,
        Enum_SiServiceStateNone             = 0x80000000,   /* may not be combined with other TEnumSiStationServiceState values */

        Enum_SiServiceStateIn               = 0x00000001,
        Enum_SiServiceStateOut              = 0x00000002,
} TEnumSiStationServiceState;

typedef struct SSiStationState
{
TEnumSiStationState          eState;        /* specifies if station is enabled/disabled */
TEnumSiStationServiceState   eService;      /* specifies if station is in service or not */
} TSSiStationState;
```

3.14 CT-Bus Routing

The ct-bus routing functions provide the application with the ability to enable or disable media routes between SSI devices and other ct-bus devices, or between SSI devices.

3.14.1 Functions

```
int si_GetXmtSlot(int iHandle, unsigned long* pulTimeSlot)
```

Inputs

int	iHandle	- station handle
unsigned long*	pulTimeSlot	- pointer to CTbus time slot

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

The function retrieves the network CTbus time slot number on which the device's channel is transmitting.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.

```
int si_Listen(int iHandle, unsigned long ulTimeSlot, TEnumSyncMode
eSyncMode)
```

Inputs

int	iHandle	- station handle
unsigned long	ulTimeSlot	- CTbus time slot to listen to
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Connects a channel to a network CTbus time slot. The function connects the receive channel on the device to an available network CTbus time slot. When the function returns, the receive channel is connected to the CTbus time slot.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.
Enum_SiErrorBadExtTs	Bad Ctbus time slot specified.

Termination Event

Enum_SiEvListenSuccess
Enum_SiEvListenFail

```
int si_UnListen(int iHandle, TEnumSyncMode eSyncMode)
```

Inputs

int	iHandle	- station handle
TEnumSyncMode	eSyncMode	- Enum_SyncMode, Enum_AsyncMode

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

The function disconnects a channel from the network CTBus time slot for a device that was connected previously using the si_Listen function.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorInvParm	Invalid eSyncMode specified.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.

Termination Event

Enum_SiEvUnlistenSuccess
Enum_SiEvUnlistenFail

3.15 Event Masking

```
int si_SetEventMask(int iHandle, const TEnumSiEventId* peEventList, int
iNumEntries, TEnumSiEventState eEventEnable)
```

Inputs

int	iHandle	- station handle
const TEnumSiEventId*	peEventList	- list of events to enable/disable
int	iNumEntries	- number of entries in pulEventList
TEnumSiEventState	eEventEnable	- enable/disable

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

This function enables and disables the reporting of asynchronous events. Only unsolicited events (not API termination events) may be masked. Events are by default disabled.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

Enum_SiErrorBadVal	Required pointer is invalid.
Enum_SiErrorInvState	Station is busy.
Enum_SiErrorFwErr	Firmware returned an error.
Enum_SiErrorTmoErr	Command timed out.

The following events are disabled by default, and may be enabled by this function:

Enum_SiEvKeyState
Enum_SiEvDeviceServiceIn
Enum_SiEvDeviceServiceOut

3.16 Event Retrieval (SRL Mode)

When the API is operating with the standard-runtime library (SRL) , then the SRL is used to retrieve events from si devices. The SRL is also used to determine the number of si devices in the system.

```
#include <silib.h>

long int si_handler(unsigned long evhandle)
{
     printf( "si_handler() called, event is 0x%x\n", sr_getevttype(evhandle));
     return( 0 );                              }
}


main()
{
   int sidev;

   /* open si channel device */
   if((si_Open(&sidev, “ssiB1C1”, Enum_SyncMode, NULL)) == -1 ){
        printf( "si_open failed\n" );
        exit( 1 );
   }

   /* Enable a handler for all events on sidev */
   if( sr_enbhdlr( sidev, EV_ANYEVT, si_handler ) == -1 ){
      printf( "Error: could not enable handler\n)
      exit( 1 );
   }
}
```

```
#include <silib.h>

long  chdev[MAXDEVS];
unsigned long  evt_handle;

main( ... )
{
   char channel_name[12];
   int ch;
   TSSiRingerState RingerState;
   int nChans;

   nChans = si_GetNumChannels("ssiB1");

   for (ch = 0; ch < nChans; ch++)
   {
      sprintf(channel_name, "ssiB1C%d", ch);
      /* Build the channel name for each channel */
      if ( (si_Open(&chdev[ch], channel_name, Enum_SyncMode, NULL) ) == -1 ) {
            printf("si_open failed\n");
            exit(1);
       }
   }

   /* turn off all ringers, then wait for the user to do something on the phone */
   RingerState.iCadence = 0;
   RingerState.iTone = 0;
   for (ch = 0; ch < nChans; ch++)
   {
       si_SetRinger(chdev[ch], &RingerState, NULL, Enum_AsyncMode);
   }

   /* This is the main loop to control the Voice hardware */
   while (FOREVER) {

        /* wait for the event */
        sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);
        process_event( evt_handle);

    }

    int process_event( ehandle)
        unsigned long ehandle;
    {
          int hStation = sr_getevtdev(ehandle);

          switch(sr_getevttype(ehandle))
            {
            case Enum_SiEvKeyState:
                  OnKeyStateCh(hStation, sr_getevtdatap(ehandle));
                  break;

            case Enum_SiEvDeviceServiceIn:
                  OnInService(hStation, sr_getevtdatap (ehandle));
```

```
        break;

 case Enum_SiEvDeviceServiceOut:
        OnOutOfService(hStation, sr_getevtdatap (ehandle));
        break;
 }

}
```

3.17 Event Retrieval (Stand-Alone Mode: Non-SRL Mode)

NOTE: The APIs presented in this section are for stand-alone mode only. They are not available when operating with SRL and Dialogic® HMP Software.

This section describes the functions used for event retrieval when the API is operating in Stand-Alone mode. In this mode, there is no SRL. Therefore, the SI API provides an event retrieval function. The function(s) described here do not exist in SRL mode.

The event retrieval functions provide the application with the ability to receive events from the station devices.

3.17.1 Functions

```
int si_WaitEvent(const int* pSiDevs, int iHandleCnt, long lTimeout, long*
plEvent, TEnumSiError* peError)
```

Inputs

const int*	pSiDevs	- array of station handles
int	iHandleCnt	- number of station handles in pSiDevs
long	lTimeout	- msecs to wait for event (0-no wait)
long*	plEvent	- pointer where to return event handle
TEnumSiError*	peError	- if SI_FAILURE return, error code returned here

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Retrieves an event from specified station devices. Caller must allocate memory for event handle. On successful return, plEvent will point to a memory location where the event handle is stored. This event handle can be used to learn additional information about the event, such as the station with which it is associated, the event type, event data, etc.

Used only in Stand-Alone mode.

Errors

Enum_SiErrorSysNotStarted	System was not started correctly.
Enum_SiErrorBadVal	Required pointer or station handle is invalid.
Enum_SiErrorInvState	Function is being called from another thread in the same process.
Enum_SiErrorSystem	Windows® error occurred.
Enum_SiErrorNoEvents	An event was not retrieved within the specified time-out.

long si_GetEvtDev(long lEvent)

Inputs

long lEvent - event handle returned by si_WaitEvent

Returns

Device handle, if successful
-1 if no current event

Description

Returns device handle associated with the current event.

Used only in Stand-Alone mode.

long si_GetEvtType(long lEvent)

Inputs

long lEvent - event handle returned by si_WaitEvent

Returns

Event type if successful
-1 if no current event

Description

Returns event type for the current event.

Used only in Stand-Alone mode.

long si_GetEvtLen(long lEvent)

Inputs

long lEvent - event handle returned by si_WaitEvent

Returns

Length of data, if successful
-1 if no current event

Description

Returns length of the variable data associated with the current event. If there is no data associated with the current event, a value of zero is returned.

Used only in Stand-Alone mode.

void* si_GetEvtDataP(long lEvent)

Inputs

long lEvent - event handle returned by si_WaitEvent

Returns

Address of variable data block, if successful
NULL if no variable data

Description

Returns the address of the variable data block associated with the current event. Use this data pointer and the event length to extract the event data. Returns NULL if there is no current event or there is no variable data associated with the current event.

Used only in Stand-Alone mode.

void* si_GetUserContext(long lEvent)

Inputs

long lEvent - event handle returned by si_WaitEvent

Returns

Address of user context,
NULL if no user context

Description

Returns the pointer to the user context that was input as a parameter to the asynchronous function. Returns non-NULL if the specified event is an asynchronous function termination event and the application passed in a pointer to a user context when the asynchronous function was invoked.

Used only in Stand-Alone mode.

```
int si_PutEvent(int iHandle, unsigned long ulEventType, long lEventLen,
const void* pvEventData, long lErrorCode)
```

Inputs

int	iHandle	- station handle
unsigned long	ulEventType	- event type ID
long	lEventLen	- number of bytes pointed to by pEventData
const void*	pvEventData	- pointer to event data to send
long	lErrorCode	- error code to be posted with message

Returns

SI_SUCCESS(0)
SI_FAILURE(-1)

Description

Allows the application to add an event to the event queue. If event data is to be passed through the pEventData parameter, the lEventLen parameter must be set to its corresponding length. The SI API makes a copy of the user's data that is pointed to by the pEventData parameter. Therefore, the caller can dispose of the event data immediately after calling the si_PutEvent() function.

Used only in Stand-Alone mode.

Errors

If this function returns SI_FAILURE(-1), use ATDV_LASTERR() to obtain error code. Refer to ATDV_LASTERR() for list of possible system error values.

3.17.2 Example

```
void stationTask(int iStation)
{
      TSSiRingerState RingState;
      TSSiIndicatorState IndState;
      long lEvent;

      while(1)
      {
            if (0 == si_WaitEvent(&iStation, 1, 1000, &lEvent))
            {
                  switch(si_GetEvtType(lEvent))
                  {
                  case Enum_SiEvKeyState:
                        OnKeyStateCh(iStation, si_GetEvtDataP(lEvent));
                        break;

                  case Enum_SiEvDeviceServiceIn:
                        OnInService(iStation, si_GetEvtDataP(lEvent));
                        break;

                  case Enum_SiEvDeviceServiceOut:
                        OnOutOfService(iStation, si_GetEvtDataP(lEvent));
                        break;
                  }
            }
      }
}
```

3.18 Event Defines

```
typedef enum
{
        Enum_SiEvInvalid                        = 0,

        Enum_SiEvStartSuccess                   = 1,

        Enum_SiEvOpenSuccess                    = 2,
        Enum_SiEvSetRingerSuccess               = 3,
        Enum_SiEvGetRingerSuccess               = 4,
        Enum_SiEvSendAlertSuccess               = 5,
        Enum_SiEvSetVolumeSuccess               = 6,
        Enum_SiEvGetVolumeSuccess               = 7,
        Enum_SiEvSetSensitivitySuccess          = 8,
        Enum_SiEvGetSensitivitySuccess          = 9,
        Enum_SiEvSetIndicatorSuccess            = 10,
        Enum_SiEvGetKeyStateSuccess             = 11,
        Enum_SiEvGetSoftKeyTextSuccess          = 12,
        Enum_SiEvSetSoftKeyTextSuccess          = 13,
        Enum_SiEvSetLocalAudioRouteSuccess      = 14,
        Enum_SiEvGetLocalAudioRouteSuccess      = 15,
        Enum_SiEvDisplayClearSuccess            = 16,
        Enum_SiEvSetDisplayActivePageSuccess    = 17,
        Enum_SiEvSetDisplayTextSuccess          = 18,
        Enum_SiEvGetDisplayTextSuccess          = 19,
        Enum_SiEvGetDisplayCursorPosSuccess     = 20,
        Enum_SiEvSetCallerIdSuccess             = 21,
        Enum_SiEvSetParmSuccess                 = 22,
        Enum_SiEvGetParmSuccess                 = 23,
        Enum_SiEvSetStationStateSuccess         = 24,
        Enum_SiEvGetStationStateSuccess         = 25,
        Enum_SiEvListenSuccess                  = 26,
        Enum_SiEvUnlistenSuccess                = 27,
        Enum_SiEvSetCallTimerSuccess            = 28,
        Enum_SiEvGetIndicatorSuccess            = 29,
        Enum_SiEvEndSuccess                     = 30,

        Enum_SiEvStartFail                      = 31,
        Enum_SiEvOpenFail                       = 32,
        Enum_SiEvSetRingerFail                  = 33,
        Enum_SiEvGetRingerFail                  = 34,
        Enum_SiEvSendAlertFail                  = 35,
        Enum_SiEvSetVolumeFail                  = 36,
        Enum_SiEvGetVolumeFail                  = 37,
        Enum_SiEvSetSensitivityFail             = 38,
        Enum_SiEvGetSensitivityFail             = 39,
        Enum_SiEvSetIndicatorFail               = 40,
        Enum_SiEvGetKeyStateFail                = 41,
        Enum_SiEvGetSoftKeyTextFail             = 42,
        Enum_SiEvSetSoftKeyTextFail             = 43,
        Enum_SiEvSetLocalAudioRouteFail         = 44,
        Enum_SiEvGetLocalAudioRouteFail         = 45,
        Enum_SiEvDisplayClearFail               = 46,
        Enum_SiEvSetDisplayActivePageFail       = 47,
        Enum_SiEvSetDisplayTextFail             = 48,
        Enum_SiEvGetDisplayTextFail             = 49,
        Enum_SiEvGetDisplayCursorPosFail        = 50,
        Enum_SiEvSetCallerIdFail                = 51,
        Enum_SiEvSetParmFail                    = 52,
        Enum_SiEvGetParmFail                    = 53,
        Enum_SiEvSetStationStateFail            = 54,
        Enum_SiEvGetStationStateFail            = 55,
        Enum_SiEvListenFail                     = 56,
        Enum_SiEvUnlistenFail                   = 57,
        Enum_SiEvSetCallTimerFail               = 58,
        Enum_SiEvGetIndicatorFail               = 59,
```

```
        Enum_SiEvEndFail                            = 60,

        Enum_SiEvStartUnsolicited                   = 61,
        Enum_SiEvKeyState                           = 62,
        Enum_SiEvDeviceServiceIn                    = 63,
        Enum_SiEvDeviceServiceOut                   = 64,
        Enum_SiEvEndUnsolicited                     = 65,

        Enum_SiEvUser                               = 66,

} TEnumSiEventId;

typedef enum
{
       Enum_SiEventStateDisable             = 0,
       Enum_SiEventStateEnable              = 1,
}TEnumSiEventState;
```

4 Error Handling

The error handling functions provide the application with the ability to retrieve more detailed error information when an SI API function returns an error code.

If a function fails, the last error code can be retrieved using the ATDV_LASTERR() function. The function ATDV_ERRMSGP() can be used to obtain a pointer to the last error.

4.1 Functions

```
long ATDV_LASTERR(int iDevHandle)
```

Inputs

iDevHandle - device handle

Returns

AT_FAILURE if an invalid device handle

Otherwise a valid error number

Description

Returns a long value that indicates the last error that occurred on this device. For Windows® only, if the error returned by ATDV_LASTERR() is Enum_SiErrorSystem, a Windows® system error has occurred.

```
char* ATDV_ERRMSGP(int iDevHandle)
```

Inputs

iDevHandle - device handle

Returns

Pointer to a string.

Description

Returns a pointer to an ASCIIZ string that describes the last error that occurred on this device.

4.2 Error Types

TEnum_SiError:

Enum_SiErrorInvLen	Input object length is invalid
Enum_SiErrorBadBrd	Board is in a bad state or has been stopped
Enum_SiErrorBadExtTs	External time slot is unsupported at current clock rate
Enum_SiErrorBadLclTs	Invalid local time slot number
Enum_SiErrorBadType	Invalid local time slot type
Enum_SiErrorBadVal	Invalid parameter
Enum_SiErrorFieldOutOfRange	A field in a structure parameter is out of range for the station device
Enum_SiErrorFwErr	Firmware returned an error
Enum_SiErrorInvBd	Invalid device handle or board index
Enum_SiErrorInvColor	Invalid color
Enum_SiErrorInvKeyId	Invalid key id
Enum_SiErrorInvIndicatorId	Invalid indicator id
Enum_SiErrorInvSubIndicator	Invalid sub-indicator
Enum_SiErrorInvIndicatorState	Invalid indicator state
Enum_SiErrorInvParm	Specified Async/Sync eSyncMode not supported
Enum_SiErrorInvName	Invalid device or board name
Enum_SiErrorInvState	Invalid state
Enum_SiErrorInvTs	Invalid time slot number specified
Enum_SiErrorNoCT	Station not connected
Enum_SiErrorNoEvents	No events available from specified station devices
Enum_SiErrorNoMem	Cannot map or allocate memory
Enum_SiErrorNotSupported	Function is not supported on specified station device
Enum_SiErrorSystem	Windows® error occurred.
Enum_SiErrorTmoErr	Timed out waiting for reply from firmware
Enum_SiErrorCallTimerEnabled	While the call timer is enabled, the display update functions return this error.
Enum_SiErrorSysNotStarted	System was not started correctly.

4.3 Additional Functions

`char* ATDV_NAMEP(int iDevHandle)`

Inputs

iDevHandle - device handle

Returns

AT_FAILURE if an invalid device handle

Otherwise a pointer to a string containing the device name.

Description

The **ATDV_NAMEP( )** function returns a pointer to an ASCIIZ string that specifies a device name, used to open the device. Part of SRL in SRL mode.

5 Use Examples

5.1 Inbound Call

#include “silib.h”

```
void ringPhone(int iStation, char* szSourceParty, int iLineIndex)
{
TSSiRingerState RingState;
TSSiIndicatorState IndState;

      /* make the station ring */
      RingState.iCadence = 1;
      RingState.iTone = 0;
      si_SetRinger(iStation, &RingState, NULL, Enum_SyncMode);

      /* blink first call appearance */
      IndState.ulIndicatorId = Enum_SiIndicatorIdLine1+iLineIndex;
      IndState.eSubIndicator = Enum_SiSubIndicatorIndicator;
      IndState.eState = Enum_SiIndicatorStateBlink;
      IndState.eColor = Enum_SiIndicatorColorGreen;
      si_SetIndicator(iStation, &IndState, NULL, Enum_SyncMode);

      /* place source party in the display “Call From: xxxx” */
      si_DisplayClear(iStation, 0, -1, NULL, Enum_SyncMode);
      si_SetDisplayText(iStation, 0, -1, -1, 0,
                                  “Call From: “,
                                  NULL,
                                  Enum_SyncMode);
      si_SetDisplayText(iStation, 0, -1, -1, 0,
                                  szSourceParty,
                                  NULL,
                                  Enum_SyncMode);
}
```

5.2 Answer Inbound Call

This function could be invoked by a button press of line 1 event from the station.

#include “silib.h”

```
void answerPhone(int iStation, char* szSourceParty, int iLineIndex)
{
TSSiRingerState RingState;
TSSiIndicatorState IndState;
TSSiVolumeDeviceState VolState;

      /* make the station not ring */
      RingState.iCadence = 0;
      RingState.iTone = 0;
      si_SetRinger(iStation, &RingState, NULL, Enum_SyncMode);

      /* light first call appearance steady */
      IndState.ulIndicatorId = Enum_SiIndicatorIdLine1+iLineIndex;
      IndState.eSubIndicator = Enum_SiSubIndicatorIndicator;
      IndState.eState = Enum_SiIndicatorStateSteady;
      IndState.eColor = Enum_SiIndicatorColorGreen;
      si_SetIndicator(iStation, &IndState, NULL, Enum_SyncMode);

      /* don’t do anything to display, it already contains source party */

      /* enable audio to the speaker and from the mic */
      si_SetLocalAudioRoute(iStation,     Enum_SiAudioDeviceSpeakerphone,
                                          Enum_SiAudioModeBidirection,
                                          NULL, Enum_SyncMode);
}
```

5.3 Outbound Call

This function could be invoked by an off-hook event from the station.

#include “silib.h”

```
void onOffHook(int iStation, int iLineIndex)
{
TSSiRingerState RingState;
TSSiIndicatorState IndState;
TSSiVolumeDeviceState VolState;

      /* make sure the station is not ringing */
      RingState.iCadence = 0;
      RingState.iTone = 0;
      si_SetRinger(iStation, &RingState, NULL, Enum_SyncMode);

      /* light first call appearance steady */
      IndState.ulIndicatorId = Enum_SiIndicatorIdLine1+iLineIndex;
      IndState.eSubIndicator = Enum_SiSubIndicatorIndicator;
      IndState.eState = Enum_SiIndicatorStateSteady;
      IndState.eColor = Enum_SiIndicatorColorGreen;
      si_SetIndicator(iStation, &IndState, NULL, Enum_SyncMode);

      /* use voice resource to play dialtone... */

      /* update display */
      si_DisplayClear(iStation, 0, -1, NULL, Enum_SyncMode);
      si_SetDisplayText(iStation, 0, -1, -1, 0,
                                  “Dial Number: “,
                                  NULL,
                                  Enum_SyncMode);


      /* enable audio to the handset */
      si_SetLocalAudioRoute(iStation,     Enum_SiAudioDeviceHandset,
                                          Enum_SiAudioModeBidirection,
                                          NULL, Enum_SyncMode);
}
```