SRL User Manual

Version 2.0

Revision History

	Date	Change
VER 2.0.0	2006-4-28	Created the document
VER 2.0.1	2009-2-9	Amended some errors in the document
VER 2.0.2	2009-5-25	Added the description of the voice mixing function
VER 2.0.3	2009-8-21	Added the description of the M3G event
VER 2.0.4	2010-5-21	Added two parameter IDs are added in the ISX_sr_default function
VER 2.0.5	2010-7-1	Amended some errors in the document
VER 2.0.6	2010-10-27	Added the description of the trunk board with high-impedance voice mixing capability

For the latest product information, please visit our website at http://www.ehangcom.com

Announcement:

This product may have design defect or wrong, and there maybe some difference to the parameters and description of this document. We are trying to make our product documents more complete and more accurate by the time it published, but because of the constant changes to actual situation, this document can only provide general information of Ehangcom products.
Ehangcom assumes no liability on the developing and marketing any particular functionally products in this document. Nor any mention or imply a commitment of product applications. This document may update anytime without notice. Therefore, it should not be treated as commitments and assurances.

Ehangcom makes no responsibility of any technical or typesetting errors and any of the resulting in this document.

Ehangcom products are not intended for use in medical, life saving, or life sustaining applications.

Brands and Intellectual Property Rights

Example

Ehangcom, iSX, iSX4000, iSX UAP,Ehcomm are trademarks of Ehangcom Corporation。

Microsoft Windows, Microsoft Windows 98, Microsoft Windows 95, Microsoft NT are registered trademarks of Microsoft Corporation.

SUN Solaris is the trademark of SUN MicroSystem Corporation.

Contents

About This Document

Welcome to this document. It is the ISX4000 SRL user manual. The software-related purpose, intended audience, document description and relevant information are as follows:

Purpose

This manual provides information about the SRL functions in SDK of iSX platform.

Intended Audience

1. Distributors
2. System Integrators
3. Toolkit Developers
4. Independent Software Vendors(ISVs)
5. Value Added Resellers(VARs)
6. Original Equipment Manufactures(OEMs)

How to Use This Manual

This manual is concomitant with the software installation. This document mainly includes the following sections:

1. Overview: This section describes the general use of this manual.

2. API Initialization Functions: This section describes how to use the API initialization and uninitialization functions.

3 Standard Extension Functions: This section describes how to use the extension functions such as obtaining API function error information.

4. Event Functions: This section describes how to use the event information obtaining functions.

Relevant Information

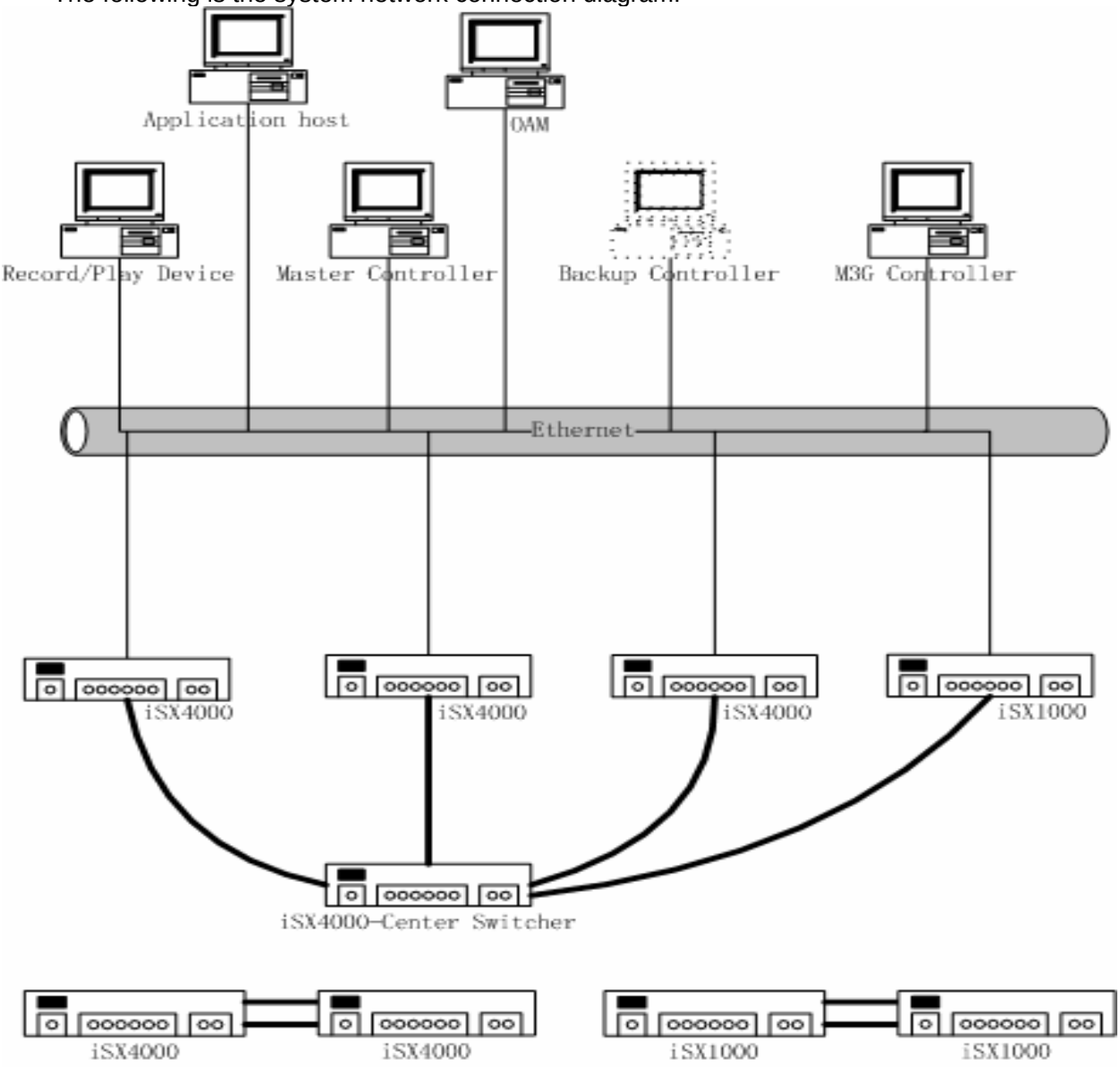
For relevant information of this manual, refer to the following documents:

1. *ISX4000 DTI User Manual*
2. *ISX4000 Global Call User Manual*
3. *ISX4000 ISX CALL User Manual*
4. *ISX4000 VOICE User Manual*
5. *ISX4000 SDK Programming Guide*
6. *OAM SDK User Manual*
7. *PRD SDK User Manual*
8. *ISX4000 Software Installing Manual*

Chapter 1 Overview

The iSX4000 system is composed of iSX4000/iSX1000 switch nodes, master controller and play/record devices. The maximal system can have 16 iSX4000/iSX1000 switch nodes, 1 center switch node, 1 master controller node and 32 play/record devices. The voice data switching between iSX4000 nodes can be implemented by the ISX4000 center switch through fiber optic couple.

The following is the system network connection diagram:



Note:

The two iSX4000 switch can be coupled by fiber optic, trunk or VoIP.
The two iSX1000 switch can be coupled by trunk or VoIP.
By now, the iSX4000 center switch and backup controller are developing.
The master controller node is a computer that has MC application running.
The Play/Record device nodes are computers that have PRD application running.

An iSX4000 node is composed of 1 motherboard, trunk daughter boards, STM-1 optiic interface daughter board ,DSP daughter boards, XOIP daughter boards, M3G daughter boards and signaling (PRI/SS7/SIP) daughter boards.

The follows are the function description of these boards:

Mother board: mainly responsible for clock management, 16K x 16K switching and daughter board power supply.

Trunk daughter board: provides a trunk interface. The interface can be configured as E1, T1 and J1. A daughter board has 8 trunks. A iSX4000 node maximal has 8 trunk daughter boards and a iSX1000 has only 1 trunk daughter boards.

STM-1 Optic interface board: For simplicity, we mapped all trunks on the optical fiber interface daughter board (STM-1 Optic interface daughter board) as virtual trunk daughter boards and virtual trunks. The virtual trunk board numbered from 8. A virtual trunk daughter board has 8 trunks. These trunks are called optical interface trunk. The usage of the virtual trunk board is as same as the trunk daughter board.

DSP daughter board: It mainly provides functions such as record/play, echo cancellation and voice conference. A daughter board has a maximum of 256 channels

XOIP daughter board: It mainly provides functions such as VOIP and Fax Over IP function.

M3G daughter board: VIDEO-3G-64 daughter board for short. This daughter board mainly provides functions such as audio and video file record and play and 3G-H324M call control. Each daughter board has a maximum of 64 channels.

Signaling daughter boards: include PRI, SS7 and SIP signaling daughter boards. A PRI or SS7 signaling daughter board have capability of up to 100 calls per second. A SIP has capability up to 20 calls per second.

The iSX4000 system is provided with a PRD (play and record device) for convenience of centralized management of voice files. Files are read from PRD during the voice play, and recorded voice is written to PRD during the record.

iSX1000 switch is similar to iSX4000 switch. To get the details of iSX4000/iSX1000 switch, please refer to i*SX4000 System Introduction Manual,iSX4000 Hardware Produce Datasheet* and other documents.

The Standard Runtime Library (SRL) provides common event handling and error analysis interfaces and other interfaces for all devices. SRL provides the following interface functions: event handling functions, event obtaining functions, SDK initialization functions, standard extension functions, device mapping functions and device group functions.

Chapter 2 Interface Function Information

2.1 API Initialization Functions

▪ *ISX_Api_SetIsxPrdNum()*

Name:	long ISX_Api_SetIsxPrdNum(iMaxIsxNum, iMaxPrdNum)	
Input parameters:	**int iMaxIsxNum**	Maximum number of ISX nodes
	int iMaxPrdNum	Maximum number of PRD nodes
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	API initialization functions	
Mode:	Synchronous	

Description

The **ISX_Api_SetIsxPrdNum ( )** function sets the maximum number of ISX/PRD nodes supported by ISX-SDK. The ISX-SDK only supports two ISX nodes and two PRD nodes by default. The function shall be called prior to ISX_Api_Init(), otherwise it will not take effect.

Parameter	Description
iMaxIsxNum	Number of ISX nodes supported by SDK. The maximum number is MAX_NODE_NUM. If the parameter value is -1, no modification is made and the default is used.
iMaxPrdNum	Number of PRD nodes supported by SDK. The maximum number is MAX_PRD_NUM. If the parameter value is -1, no modification is made and the default is used.

Cautions

None.

Example

```
#include <srllib.h>
main()
{
    ISX_Api_SetIsxPrdNum (4, 4);
    ISX_Api_Init(MC_ADDR, MC_PORT, KEEP_ALIVE);
    ……
}
```

Relevant information

None.

▪ *ISX_Api_Init()*

Name:	long ISX_Api_Init(szMCIp, usMCPort, ucKeepAlive)	
Input parameters:	**char***	Master controller (MC) IP address

	unsigned short usMCPort	Master controller port
	unsigned char ucKeepAlive	Checks the cycle of keep-alive messages
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	API initialization functions	
Mode:	Synchronous	

Description

The **ISX_Api_Init( )** function initializes ISX API function library that contains functions such as Voice, DTI, SRL and GC functions. Before these functions are used, the application must call **ISX_Api_Init( )** to initialize the function library first. The function is generally called during the initialization of the whole application. **ISX_Api_Uninit( )** is called at the end of the whole application. **ISX_Api_Init( )** and **ISX_Api_Uninit( )** cannot be repeatedly called in an application.

Parameter	Description
szMCIp	Master controller IP address
usMCPort	Master controller port
ucKeepAlive	Checks the cycle of keep-alive messages (i.e. heartbeat cycle). ISX API communicates with MC through TCP/IP. ISX API regularly sends keep-alive messages to MC to check whether the peer or network is normal. After receiving a keep-alive message, MC sends a response message to ISX API. If MC does not receive any keep-alive message sent by ISX API, MC automatically disconnects from ISX API. The mechanism is inconvenient for the program single-step follow-up and commissioning. Therefore, we design the parameter so that the user can set an inspection cycle. The default cycle is 20s. Values specified by the parameter are the multiple of the default cycle. For example, if usKeepAlive is equal to 3, the inspection cycle is 60s.

Cautions

None.

Errors

If the function call fails, the return value is less than 0.

-1: Parameter error. The szMCIp parameter may be empty, or the usMCPort parameter may be 0;

Other negatives: Please contact the developer.

Example

```
#include <srllib.h>
main()
{
    /* Initialize ISX API */
    if(ISX_Api_Init("192.168.30.0", 9000, 10) == -1 ){
        printf( "Error: cannot initialize ISX API\n" );
        exit( 1 );
    }
   /*Continue process    */
   /* Uninitialize ISX API */
    if(ISX_Api_Uninit() == -1 ){
        printf( "Error: cannot uninitialize ISX API\n" );
        exit( 1 );
```

```
        }
    }
```

Relevant information

- **ISX_Api_Uninit()**

▪ *ISX_Api_Uninit()*

Name:	long ISX_Api_Uninit()
Input parameters:	None
Return value:	0 success -1 failure
Header file:	srllib.h
Category:	API initialization functions
Mode:	Synchronous

Description

The **ISX_Api_Uninit()** function stops the function library from working, disconnects from MC and releases resources occupied by the function library.

Cautions

None.

Errors

If the function call fails, -1 is returned. Please view log files to determine failure causes.

Example

Refer to the program example of **ISX_Api_Init()**.

Relevant information

- **ISX_Api_Init()**

2.2 Standard Attribute Functions

▪ *ISX_ATDV_ERRMSGP()*

Name:	char * ISX_ATDV_ERRMSGP(dev)	
Input parameters:	**int dev**	Valid device handle
Return value:	Pointer to string	
Header file:	srllib.h	
Category:	Standard attribute functions	
Mode:	Synchronous	

Description

The **ISX_ATDV_ERRMSGP( )** function returns an ASCIIZ string that describes the error that occurs during the last function call. The pointer remains valid throughout the execution of the application. If no error occurrs, the function returns a “No Error” string.

Parameter	Description
dev	Valid device handle returned by the **xx_open()** function

Cautions

None.

Errors

If the function specifies an invalid device handle, it returns an “Unknown device” string.

Example

```
#include <srllib.h>
#include <voclib.h>
main()
{
    int chdev;
   /* Open voice channel device */
    if(( chdev = ISX_dx_open(DT_DSP_CH, 0, 0, 0)) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
   /*Attempt to play a file on a channel and device-will fail */
    if( ISX_dx_play( chdev, ...) == -1 ){
        printf( "The last error on the device: '%s'\n",ISX_ATDV_ERRMSGP(chdev));
    }
}
```

Relevant information

None.

▪ *ISX_ATDV_LASTERR()*

Name:	long ISX_ATDV_LASTERR(dev)	
Input	**int dev**	Valid device handle

parameters:		
Return value:	AT_FAILURE: failure. The specified handle is invalid Others: error code	
Header file:	srllib.h	
Category:	Standard attribute functions	
Mode:	Synchronous	

Description

The **ISX_ATDV_LASTERR()** function returns codes (long integers) that indicates errors that occurr during the last function call. The error codes are located in all relevant header files such as voclib.h and dtilib.h.

Parameter	Description
dev	Valid device handle returned by the **ISX_xx_open()** function

Cautions

None.

Errors

If an invalid device handle is specified, the function returns AT_FAILURE.

Example

```
#include <srllib.h>
#include <voclib.h>
main()
{
    int chdev;
   /* Open voice channel device */
    if(( chdev = ISX_dx_open(DT_DSP_CH, 0, 0, 0)) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
   /*Attempt to play a file on a channel and device-will fail */
    if( ISX_dx_play( chdev, ...) == -1 ){
        printf( "The last error on the device: %d\n",ISX_ATDV_LASTERR(chdev));
    }
}
```

Relevant information

None.

▪ *ISX_sr_getlasterr()*

Name:	long ISX_sr_getlasterr()
Input parameters:	None
Return value:	-1: failure Others: error code during the last function call
Header file:	srllib.h
Category:	Standard attribute functions
Mode:	Synchronous

Description

The **ISX_sr_getlasterr()** function obtains the error code during the last call function. The function is similar to the **GetLastError()** function in Windows API. Their difference is that **ISX_ATDV_LASTERR()** applies to devices and **ISX_sr_getlasterr()** applies to threads. **ISX_sr_getlasterr()** is generally used to obtain failure causes of the ISX_xx_open() function.

Cautions

None.

Errors

If the function call fails, -1 is returned.

Example

Refer to the program example of **ISX_dx_open()**.

Relevant information

- **ISX_dx_open()**
- **ISX_dx_close()**

2.3 Event Functions

▪ *ISX_sr_waitevt()*

Name:	long ISX_sr_waitevt(timeout)	
Input parameters:	**long timeout**	Timeout time in ms
Return value:	0 success.An event occurs -1 failure. No event occurs	
Header file:	srllib.h	
Category:	Event handling functions	
Mode:	Synchronous	

Description

The **ISX_sr_waitevt ()** function waits for an event within the specified time. If an event occurs, 0 is returned immediately. If no event occurs within the specified time, -1 (SR_TMOUT) is returned.

Parameter	Description
timeout	Timeout time in ms. If the parameter value is -1, it indicates infinite waiting.

Cautions

- When an event is received, the event must be handled immediately before the **ISX_sr_waitevt()** function is called again. This is because **ISX_sr_waitevt()** automatically clears the current event before waiting for a new event.
- **Unix:** Unlike other companies, we do not use **time()** to achieve **ISX_sr_waitevt( )**. Therefore, the time precision can achieve millisecond level, not one-second level.
- **ISX_sr_waitevt()** and **ISX_sr_waitevtEx()** shall not be used in the same application.

Errors

If no event occurs during the specified time and the function times out, the function returns -1 (SR_TMOUT).

Example

```
#include <srllib.h>
main()
{
    /* Wait 10 seconds for an event */
    if( ISX_sr_waitevt( 10000 ) == -1 ){
        printf( "ISX_sr_waitevt time out! \n");
    }
    else{
            printf( "Got event 0x%x on device %d, data len = %d, datap = 0x%x\n",
                ISX_sr_getevttype(),ISX_sr_getevtdev(), ISX_sr_getevtlen(),
                ISX_sr_getevtdatap());
    }
}
```

Relevant information

- **ISX_sr_waitevtEx()**

▪ *ISX_sr_waitevtEx()*

Name:	long ISX_sr_waitevtEx(handlep, handlecnt, timeout, rfup)	
Input parameters:	**long *handlep**	Handle array address of the device that needs to wait for an event
	int handlecnt	Number of devices contained in the Handlep array
	long timeout	Timeout time in ms
	long *rfup	Reserved for future use
Return value:	0 success -1 timeout	
Header file:	srllib.h	
Category:	Event handling function	
Mode:	Synchronous	

Description

The **ISX_sr_waitevtEx( )** function waits for device events. The function can be regard as an extension of the **ISX_sr_waitevt()** function. If events are detected on the specified devices, the function returns 0 immediately. If no event occurs on the specified devices within the specified time, the function times out and returns -1 (SR_TMOUT).

Parameter	Description
handlep	Device handle array
handlecnt	Number of handles in the handle array
timeout	Timeout time in ms. If the value is -1, it indicates infinite waiting until an event occurs.
rfup	Reserved for future use

Cautions

- The **ISX_sr_waitevtEx()** function may be used in a single thread or in multiple threads. However, the same device cannot be simultaneously used in various **ISX_sr_waitevtEx()** functions, otherwise unexpected results may occur. In addition, events must be handled in the same thread in which the **ISX_sr_waitevtEx()** function is called.
- If the function waits for system events, the device handle shall be set to SR_SYSHDL.
- **ISX_sr_waitevt()** and **ISX_sr_waitevtEx( )** shall not be used in the same application.

Errors

If no event occurs on the specified devices within the specified time, the function times out and returns -1 (SR_TMOUT).

Example

```
#include <srllib.h>
#include <voclib.h>
```

```
long    chdev[MAXDEVS+1];
int process_event()
{
   long varlen;
   int     iOperType;
   void* pVarData;
   int    voxhandle = ISX_sr_getevtdev();
   switch(ISX_sr_getevttype()) {
       case TDX_RECORD:
           varlen = ISX_sr_getevtlen();
           pVarData = ISX_sr_getevtdatap();
              .
           break;
       case TDX_PLAY:
              .
           break;
       case TDX_ERROR:
           iOperType = ISX_sr_getopertype();

           if(iOperType == OPER_PLAY) printf("Play error!\n");
           else if(iOperType == OPER_RECORD) printf("record error!\n");
           else if(...) …;

           break;
   }
}
main( ... )
{
    int ch;
    for (ch = 0; ch < MAXDEVS; ch++) {
          if ( (chdev[ch] = ISX_dx_open(DT_DSP_CH, 0, 0, i) ) == -1 ) {
             printf("ISX_dx_open failed\n");
             exit(1);
          }
    }
    chdev[MAX_DEVS] = SR_SYSHDL;
   /*
    * Now initialize each device and then issue the command asynchronously to
    * start off the state machine.
    */
   /* This is the main loop to control the Voice hardware */
    while (FOREVER) {
       /* wait for the event */
        if(ISX_sr_waitevtEx( chdev, MAXDEVS, -1, NULL) == 0){
             process_event();
        }
    }
}
```

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_getevtdev()**
- **ISX_sr_getevttype()**
- **ISX_sr_getevtlen()**
- **ISX_sr_getevtdatap()**

▪ *ISX_sr_getevtdev()*

Name:	long ISX_sr_getevtdev()
Input parameters:	None
Return value:	-1: failure Others: event device handle
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getevtdev( )** function returns the device handle of the current event. If no current event exists, -1 is returned. If timeout occurs when waiting for an event, the function returns SRL_DEVICE. If SR_SYSHDL is returned, it indicates that the event is a system event.

Cautions

None.

Errors

None.

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**

▪ *ISX_sr_getevttype()*

Name:	long ISX_sr_getevttype( )
Input parameters:	None
Return value:	-1 failure Others: event type
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getevttype( )** function returns the type of the current event. If no current event exists, -1 is returned. If timeout occurs while waiting for an event, the function returns SR_TMOUTEVT.

Cautions

None.

Errors

None

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**

▪ *ISX_sr_getevtlen()*

Name:	long ISX_sr_getevtlen( )
Input parameters:	None
Return value:	-1 failure Others: event length
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getevtlen( )** function returns the length of the additonal data of the current event. If no current event exists, 0 is returned.

Cautions

None.

Errors

None.

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**

▪ *ISX_sr_getevtopertype()*

Name:	int ISX_sr_getevtopertype( )
Input parameters:	None
Return value:	Returns the event-related operation type
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getopertype( )** function returns the event-related operation type. Operation types include the following:

OPER_UNKNOWN: Unknown operation type;
OPER_PLAY: Playback operation;
OPER_RECORD: Recording operation;

OPER_GETDIG: Get-digit operation;
OPER_ACCEPTCALL: ISX_gc_AcceptCall() operation;
OPER_ANSWERCALL: ISX_gc_AnswerCall() operation;
OPER_MAKECALL: ISX_gc_MakeCall() operation;
OPER_DROPCALL: ISX_gc_DropCall() operation;
OPER_SETBILLING: ISX_gc_SetBilling() operation;
OPER_SENDMOREINFO: ISX_gc_SendMoreInfo() operation;
OPER_DLCACHE: Cache voice downloading operation;
OPER_DLMEM: Memory voice downloading operation;
OPER_STOPCH: Channel I/O stopping operation;
OPER_PUTEVT: For this type of events, the user calls the ISX_sr_putevt() function to insert the user events into an event queue;
OPER_SYSNOTIFY: System notification event;
OPER_CST: CST event detected by the system
OPER_GETTRCOUNT: Operation for obtaining the number of played or recorded bytes;
OPER_XOIP_SETRESATTR: ISX_ipm_SetUsrAttr() operation；
OPER_XOIP_GETRESATTR: ISX_ipm_GetUsrAttr() operation；
OPER_XOIP_GETSSINFO: ISX_ipm_GetSessionInfo() operation；
OPER_XOIP_STARTMEDIA: ISX_ipm_StartMedia() operation；
OPER_XOIP_STOP: ISX_ipm_Stop() operation；
OPER_XOIP_SENDDIGITS: ISX_ipm_SendDigits() operation；
OPER_XOIP_ENABLEEVENT: ISX_ipm_EnableEvents () operation；
OPER_XOIP_DISABLEEVENT: ISX_ipm_DisableEvents () operation;
OPER_XOIP_RECEIVEDIGITS: ISX_ipm_ReceiveDigits () operaton；
OPER_XOIP_SENDRFC2833: ISX_ipm_SendRFC2833SignalIDToIP () operation；
OPER_SENDCPG: ISX_gc_SendCPG () operation；
OPER_CALLPROCEEDING: ISX_gc_SendCallProceeding ()operation；
OPER_CALLPROGRESS: ISX_gc_CallProgress () operation；
OPER_SETUPACK: ISX_gc_SetupAck () operation；
OPER_SIP_SENDREINVITE: ISX_gc_SIPSendReinvite() operation;
OPER_SIP_SENDREINVITEACK: ISX_gc_SIPSendReinviteAck() operation;
OPER_SIP_SENDREGISTER: ISX_gc_SIPSendRegister() operation;
OPER_XOIP_SWITCH_VF: ISX_ipm_SwitchVF() operation;
OPER_SIP_SENDINFO: ISX_gc_SIPSendInfo() operation;
OPER_SIP_SENDINFOACK: ISX_gc_SIPSendInfoAck() operation;
OPER_NR_SCROUTE: ISX_nr_scroute() operation;
OPER_NR_SCUNROUTE: ISX_nr_scunroute() operation;
OPER_SIP_SENDREGISTERACK: ISX_gc_SIPSendRegisterAck() operation;
OPER_SIP_SENDTRYING: ISX_gc_SIPSendTrying() operation

Cautions

None.

Errors

None.

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**
- **ISX_sr_getevtlen()**

ISX_sr_getevtoperindex()

Name:	int ISX_sr_getevtoperindex( )
Input parameters:	None
Return value:	Returns the number of the operation corresponding to the event
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getevtoperindex ( )** function returns the event-related operation number. You can specify an operation index for some operations such as recording, playback and obtaining the number of played or recorded bytes. When an event is generated, the number can be obtained through ISX_sr_getevtoperindex() so that a programmer can know which operation results in the event.

Cautions

None.

Errors

None.

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**
- **ISX_sr_getevtlen()**

ISX_sr_getevtdatap()

Name:	void* ISX_sr_getevtdatap( )
Input parameters:	None
Return value:	NULL: no variable-length data for the event Others: address of variable-length data of the event
Header file:	srllib.h
Category:	Event information obtaining functions
Mode:	Synchronous

Description

The **ISX_sr_getevtdatap( )** function returns the address of the additional data of the current event. Please use **ISX_sr_getevtlen()** to determine the valid length of the variable-length data. If there is no variable-length data for the current event, NULL is returned. The returned pointer is valid before **ISX_sr_waitevt()** or **ISX_sr_waitevtEx()** is called again.

Cautions

None.

Errors

None.

Example

Refer to the example of **ISX_sr_waitevtEx()**.

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**
- **ISX_sr_getevtlen()**

▪ *ISX_sr_getevtname()*

Name:	char * ISX_sr_getevtname(iEvtType)	
Input parameters:	**int iEvtType**	Event type
Return value:	Pointer to string	
Header file:	srllib.h	
Category:	Event handling functions	
Mode:	Synchronous	

Description

The **ISX_sr_getevtname( )** function obtains the event name of a certain event or the current event (iEvtType is SR_CUREVENT).

Parameter	Description
iEvtType	Event type

Cautions

None.

Example

```
#include <srllib.h>
#include <voclib.h>
main()
{
    Int iEvtType = ISX_sr_getevttype()
    printf( "Get :'%s'\n",ISX_sr_getevtname(iEvtType));
}
```

Relevant information

- None.

▪ *ISX_sr_putevt()*

Name:	long ISX_sr_putevt(dev, evttype, evtlen, evtdatap, errcode)	
Input parameters:	**long dev**	Device handle
	unsigned long	Event type

	evttype	
	long evtlen	Variable-length data length
	void *evtdatap	Pointer to variable-length data
	long errcode	Error code
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Event handling functions	
Mode:	Synchronous	

Description

The **ISX_sr_putevt( )** function allows the application to add an event in the ISX API event queue. If there is variable-length data, please use **evtdatap** to specify the source data address and specify the length of the variable-length data through the **evtlen** parameter. The **ISX_sr_putevt( )** function will copy the variable-length data to the queue. Therefore, the user may discard **evtdatap** when the function returns.

Parameter	Description
dev	Device handle
evttype	Event type. It may be a user-defined type. The value can be obtained through **ISX_sr_getevttype()**.
evtlen	Variable-length event length. The value can be obtained through **ISX_sr_getevtlen()**. If there is no variable-length data, please set the parameter to 0.
evtdatap	Variable-length data address. Variable-length data can be obtained through **ISX_sr_getevtdatap()**. If there is no variable-length data, please set the parameter to NULL.
errcode	Error code. The value can be obtained through **ISX_ATDV_LASTERR()**. If the value is 0, no error code is set.

The event generated by the **ISX_sr_putevt()** function can be obtained through the **ISX_sr_waitevt()** or **ISX_sr_waitevtEx()** function.

Cautions

- If an invalid device handle is specified, the function returns failure.

Errors

None.

Example

```
#include <srllib.h>
#include <voclib.h>
int             dev;        /* voice device handle */
/* Open voice device */
if ((dev = ISX_dx_open(DT_DSP_CH, 0, 0, 0)) == -1) {
    printf("Cannot open voice channel");
    exit(1);
}
/* Put an play complete event on the event queue */
```

```
if (ISX_sr_putevt(dev, TDX_PLAY, 0, NULL, 0) == -1){
    …
}
```

Relevant information

- **ISX_sr_waitevt()**
- **ISX_sr_waitevtEx()**

2.4 Utility Tool Functions

▪ *ISX_sr_default()*

Name:	long ISX_sr_default(iParmId, pParam)	
Input parameters:	**int iParmId**	Parameter ID
	void* pParam	Parameter address
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

Some parameters are very complex in the functions provided by ISX SDK, especially in the Global Call functions. For the convenience of programming by the user, we provide the ISX_sr_default() function to set some complex parameters to the defaults.

Parameter	Description
iParmId	Parameter ID
pParam	Parameter address

Parameter ID list

iParmId	Pointer to the Structure Corresponding to pParam
PARMID_SS7_IAM	SS7_IAM
PARMID_SS7_ACM	SS7_ACM
PARMID_SS7_ANM	SS7_ANM
PARMID_SS7_REL	SS7_DROP
PARMID_SS7_CPG	SS7_CPG
PARMID_SIP_INVITE	SIP_INVITE
PARMID_PRI_SETUP	PRI_SETUP
PARMID_PRI_ALERTING	PRI_ALERTING
PARMID_PRI_CONNECT	PRI_CONNECT
PARMID_PRI_DROP	PRI_DROP
PARMID_PRI_SETUPACK	PRI_SETUPACK

PARMID_PRI_PROGRESS	PRI_PROGRESS
PARMID_PRI_PROCEEDING	PRI_CALLPROCEEDING
PARMID_SIP_INVITE_EX_REQ	SIP_INVITE_EX
PARMID_SIP_OPTIONS	SIP_OPTIONS

Cautions

None.

Errors

None.

Example

```
#include <srllib.h>
#include <gclib.h>

/* Set IAM to default */
GCPARAMEX_MAKECALL MakeCallEx;
MakeCallEx.Protocol = eGCPro_SS7;
ISX_sr_default(PARMID_SS7_IAM, &MakeCallEx.u.SS7Iam);
strcpy(MakeCallEx.u.SS7Iam.CGPN.calling_number, "888");
strcpy(MakeCallEx.u.SS7Iam.CDPN.called_number, "999");
iRetVal = ISX_gc_MakeCall(linedev, &crn, NULL, NULL, -1, EV_ASYNC |
                usOverlapMode, &MakeCallEx);
...
}
```

Relevant information

None.

▪ *ISX_sr_SetLogDir()*

Name:	long ISX_sr_SetLogDir(pszLogPath)	
Input parameters:	**char* pszLogPath**	Log saving path
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_SetLogDir( )** function sets a path for logs output by SDK. The **ISX_sr_SetLogFilter()** function specifies whether to save logs output by SDK to files. File names that save SDK logs are determined by SDK (currently, another file name shall be used every 500,000 lines and the file naming rule is "*LogPath* ISdkLogyyyymmdd_hhnnss.txt", i.e. "*LogPath* ISdkLog(year)(month)(day)_(hour)(minute)(second).txt").

Parameter	Description
pszLogPath	Log saving path

Cautions

None.

Example

```
#include <srllib.h>
#include <voclib.h>
main()
{
    ISX_sr_SetLogDir("c:\\Log\\");
    LOG_FILTER Filter;
    memset(&Filter, 0, sizeof(Filter));
    Filter.ucLog2File          = TRUE;
    Filter.ucOutputErrLog      = TRUE;
    Filter.ucOutputSYSLog      = TRUE;
    Filter.ucOutputGCLog       = TRUE;
    Filter.ucOutputDSPLog      = TRUE;
    Filter.ucOutputXOIPLog     = TRUE;
    Filter.ucOutputDTILog      = TRUE;
    Filter.ucOutputCSLog       = TRUE;
    Filter.ucOutputPRDLog      = TRUE;
    ISX_sr_SetLogFilter(&Filter);
    ISX_sr_StartLogServer(8181);

    Sleep(2000);
    ISX_Api_Init(szMCAddr, iPort, KEEP_ALIVE);
    ……
}
```

Relevant information

None.

▪ *ISX_sr_SetLogFilter()*

Name:	long ISX_sr_SetLogFilter(pFilter)	
Input parameters:	**LOG_FILTER* pFilter**	Log filtering condition
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_SetLogFilter( )** function sets filtering conditions for SDK output logs.

Parameter	Description
pFilter	Log filtering conditions. Log conditions are defined by the LOG_FILTER structure. Each field is described as follows: **ucLog2File:** Whether to save to files. **ucOutputErrLogOnly:** Outputs error logs only. If the field is TRUE, ucOutputErrLog, ucOutputSYSLog, ucOutputGCLog, ucOutputDSPLog, ucOutputXOIPLog, ucOutputDTILog, ucOutputCSLog and ucOutputPRDLog are not analyzed. **ucOutputErrLog:** Whether to output error logs. If the field is TRUE, error logs are output, otherwise only normal logs are output. **ucOutputSYSLog:** Whether to output system logs. If the field is TRUE, system logs

	are output, otherwise no system logs are output. **ucOutputGCLog**: Whether to output GC logs. If the field is TRUE, GC logs are output, otherwise no GC logs are output. GC logs are mainly signaling logs including PRI/SS7/SIP signaling function-related logs. **ucOutputDSPLog**: Whether to output DSP logs. If the field is TRUE, DSP logs are output, otherwise no DSP logs are output. DSP logs are mainly logs related to functions such as playback, recording, conferencing and DTMF functions. **ucOutputXOIPLog**: Whether to output XOIP logs. If the field is TRUE, XOIP logs are output, otherwise no XOIP logs are output. XOIP logs are mainly logs related to VOIP/FOIP media functions. **ucOutputDTILog**: Whether to output DTI logs. If the field is TRUE, DTI logs are output, otherwise no DTI logs are output. DTI logs are mainly logs related to DTI (Digital Trunk Interface) functions. **ucOutputPRDLog**: Whether to output fileopen/fileclose logs. If the field is TRUE, fileopen/fileclose logs are output, otherwise no fileopen/fileclose logs are output. **ucDbgIdFilterValid:** Whether to enable the DbgId condition filtering. If the field is TRUE, channel DbgId conforms to the condition settings in DbgIdFilter besides the above-mentioned conditions. Channel DbgId is set by ISX_sr_SetDbgIdStr(). **usDbgIdNum**: Number of DbgId filtering conditions. **DbgIdFilter:** DbgId filtering condition array. Only the logs that satisfy one of the DbgId filtering condition array. The maximum number of conditions in the DbgId filtering condition array is MAX_DBGID_NUM. Each filtering condition has two fields. One field is ucMode that specifies a DbgId string comparison mode. 0 indicates perfect matching and 1 indicates containing matching. If the top digit of ucMode is 1, it indicates ignoring case.

Cautions

None.

Example

```
#include <srllib.h>
#include <voclib.h>
main()
{
    ISX_sr_SetLogDir("c:\\Log\\");

    LOG_FILTER Filter;
    memset(&Filter, 0, sizeof(Filter));
    Filter.ucLog2File           = TRUE;
    Filter.ucOutputErrLog               = TRUE;
    Filter.ucOutputSYSLog               = TRUE;
    Filter.ucOutputGCLog                = TRUE;
    Filter.ucOutputDSPLog               = TRUE;
    Filter.ucOutputXOIPLog              = TRUE;
    Filter.ucOutputDTILog               = TRUE;
    Filter.ucOutputPRDLog               = TRUE;

    Filter.ucDbgIdFilterValid   = TRUE;
    Filter.usDbgIdNum           = 1;
    Filter.DbgIdFilter[0].ucMode  = 0;
    strcpy(Filter.DbgIdFilter[0].szDbgIdStr, "38664160→38668899");

    ISX_sr_SetLogFilter(&Filter);
    ISX_sr_StartLogServer(8181);

    Sleep(2000);
    ISX_Api_Init(szMCAddr, iPort, KEEP_ALIVE);
    ……

    int iDspH = ISX_dx_open(DT_DSP_CH, 0, 0, 0, NULL);
```

```
    ISX_sr_SetDbgIdStr(iDspH, "38664160→38668899");
    ISX_dx_close(iDspH);//Logs can be output because the DbgId of the channel is the same as the DbgId in the filtering conditions

    int iDtiH = ISX_dt_open(0, 0, 0, 0);
    ISX_sr_SetDbgIdStr(iDtiH, "12345678→87654321");
    ISX_dt_close(iDtiH);// Logs cannot be output because the DbgId of the channel is different from the DbgId in the filtering conditions

    ……
}
```

Relevant information

None.

▪ *ISX_sr_StartLogServer()*

Name:	long ISX_sr_StartLogServer(iPort)	
Input parameters:	**int iPort**	LogServer port
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_StartLogServer( )** function starts up LogServer. SDK logs can be saved to files. They can also be connected to LogServer through TCP/IP by using IsxLog App so that SDK output logs are transmitted to IsxLog App for display in real time. Each APP that calls SDK must use independent LogServer, which means that when multiple APPs (programs calling ISX API) run in the same computer, LogServer must be started up by using various ports, otherwise it will result in port conflict.

Parameter	Description
iPort	LogServer port

Cautions

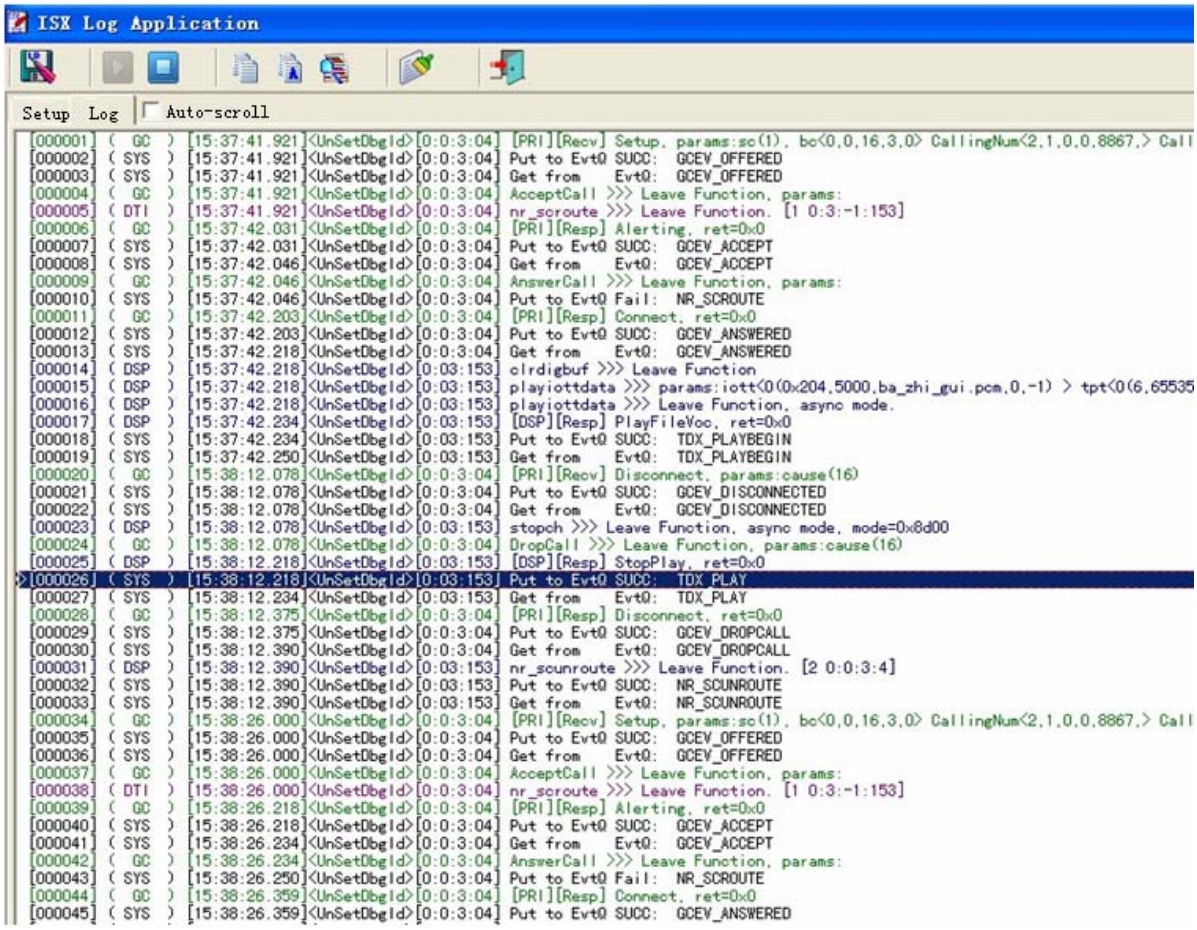
None.

Example

Refer to the examples of **ISX_sr_SetLogDir()** and **ISX_sr_StartLogServer()**.

Relevant information

The following figure shows the screenshot of IsxLog-App:



▪ *ISX_sr_SetDbgIdStr()*

Name:	long ISX_sr_SetDbgIdStr(iDev, szIdStr)	
Input parameters:	**int iDev**	Device handle of DbgId to be set
	char* szIdStr	DbgId string
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_SetDbgIdStr( )** function sets channel DbgId that is a string. If DbgId filtering conditions are set during the log output, the If the DbgId in the DbgId filtering conditions are compared with the DbgId of the channel. If the two strings are equal (or have a inclusion relation, the comparison mode is specified in the DbgId filtering conditions), logs are output, otherwise no logs are output.

Parameter	Description

iDev	Device handle of DbgId to be set. It may be a channel handle such as DTI, DSP, GC and CS channel handles.
szIdStr	DbgId string. The maximum length shall not exceed MAX_DBGID_LEN-1.

Cautions

None.

Example

Refer to the examples of **ISX_sr_SetLogDir()** and **ISX_sr_StartLogServer()**.

Relevant information

None.

ISX_sr_GetSdkVer()

Name:	char* ISX_sr_GetSdkVer()
Input parameters:	None
Return value:	0 success -1 failure
Header file:	srllib.h
Category:	Utility tool function
Mode:	Synchronous

Description

The **ISX_sr_GetSdkVer( )** function obtains the SDK version number. The version number returned by the function is the version number of the SDK DLL. The user can compare this version with ISX_API_VER in the srllib.h header file. If they are equal, it indicates that the header file used during the SDK DLL compilation is the same as the header file used during the APP compilation.

Parameter	Description
None	

Cautions

None.

Example

```
#include <srllib.h>
main()
{
    char* pVer = ISX_sr_GetSdkVer();
    if(strcmp(pVer, ISX_API_VER)){
        printf("WARNING : wrong sdk version! DLL Ver:%s, H_Ver: %s\n",
                pVer, ISX_API_VER);
    }
    else{
        printf("DLL ver: %s\n", pVer);
    }

    ……
}
```

Relevant information

None.

▪ *ISX_sr_TrunkChMapping()*

Name:	long ISX_sr_TrunkChMapping (bEnable)	
Input parameters:	**BOOL bEnable**	Whether to use the mapping mode
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

If the type of a trunk is E1, the DTI and GC channels of the trunk have two numbering modes, i.e. mapping mode and non-mapping mode. For the mapping and non-mapping modes, refer to the description of the ISX_dt_open() and ISX_gc_OpenEx(). The mode must be specified by calling the **ISX_sr_TrunkChMapping()** function. The function must be called before the ISX_dt_open() and ISX_gc_OpenEx() functions are called. After the ISX_dt_open() or ISX_gc_OpenEx() function is called, **ISX_sr_TrunkChMapping()** shall not be called, otherwise it will cause confusion. The default is mapping mode.

Parameter	Description
bEnable	Whether to use the mapping mode. If it is TRUE, the mapping mode is used. If it is FALSE, the non-mapping mode is used.

Cautions

None.

Example

```
#include <srllib.h>
main()
{
    ISX_sr_TrunkChMapping(FALSE);

    /* Open dti channel*/
    /* Open gc channel */

    ……
}
```

Relevant information

None.

▪ *ISX_sr_getdevtype()*

Name:	long ISX_sr_getdevtype(dev)	
Input parameters:	**long dev**	Device handle
Return value:	0 success -1 failure	
Header file:	srllib.h	
Category:	Utility tool functions	

Mode:	Synchronous

Description

The **ISX_sr_getdevtype()** function obtains the type of a specified device. Device types are as follows:

```
#define DT_DSP_BRD      0       //DSP daughter board
#define DT_DSP_CH  1        //DSP channel
#define DT_DTI_CH   2       //Trunk channel
#define DT_SIG_CH   3       // SS7 & PRI (Global Call) line channel
#define DT_CS_CH    4       //CS channel (for CS switching)
#define DT_XOIP_CH 5        //XOIP channel
#define DT_PRD      6       // PRD
#define DT_SIP_CH   7       // SIP (Global Call) line device
#define DT_SYS      127     // System device
```

Parameter	Description
Dev	Device handle

Cautions

None.

Example

```
#include <srllib.h>
main()
{
    char DevTypeDesc[][32]={
    “DT_DSP_BRD”,
    “DT_DSP_CH”,
    “DT_DTI_CH”,
    “DT_SIG_CH”,
    “DT_CS_CH”,
    “DT_XOIP_CH”,
    “DT_PRD”,
    “DT_SIP_CH”,
    “DT_SYS”,
    }

    int dev = ISX_dt_open(…);
    long devtype = ISX_sr_getdevtype(dev);
    if(devtype >= 0)
        printf(“The type of dev(%d) is: %s\n”, DevTypeDesc[devtype]);
    else
        printf(“ISX_sr_getdevtype fail.\n”);

    ……
}
```

Relevant information

None.

▪ *ISX_sr_getnet2cfg()*

Name:	long ISX_sr_getnet2cfg(ucIsxNo, ucBrdNo, ucBrdType, ipp, netmaskp)	
Input parameters:	**unsigned char ucIsxNo**	Node number
	unsigned char	Daughter board number

	ucBrdNo	
	unsigned char ucBrdType	Daughter board type
	char* ipp	Pointer to IP address
	char* netmaskp	Pointer to mask address
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

Many daughter boards in the ISX4000 switch have two network ports. The IP address of the first network port is internally assigned (automatically calculated according to the IP address of the motherboard). The IP address of the second network port is set by the user. The user can set the IP address of the second network port through OAM. The **ISX_sr_getnet2cfg()** function obtains the IP address information of the second network port.

Parameter	Description
ucIsxNo	Node number
ucBrdNo	Board number
ucBrdType	Board type as follows: BT_DSP: DSP daughter board BT_XOIP: XOIP daughter board BT_SIP: SIP daughter board
ipp	Pointer to IP address
netmaskp	Pointer to mask address

Cautions

None.

Errors

None.

Example

```
#include <srllib.h>
#include <ipmlib.h>

char ip[32];
char netmask[32];
ISX_sr_getnet2cfg(ucIsxNo, ucBrdNo, BT_XOIP, ip, netmask);
printf("IPM PORT2, ip:%s, netmask:%s\n", ip, netmask);
```

Relevant information

None.

▪ *ISX_sr_GetSpecCap()*

Name:	long ISX_sr_GetSpecCap(ucIsxNo, ucBrdNo, ucBrdType, pSpecCap)	
Input parameters:	**unsigned char ucIsxNo**	Node number
	unsigned char ucBrdNo	Daughter board number
	unsigned char ucBrdType	Daughter board type
	SPEC_CAP *pSpecCap	Pointer to the address where special capabilities are stored
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_GetSpecCap** function obtains special capabilities of daughter boards. Some daughter boards of ISX series have some special capabilities. Currently, the trunk voice mixing daughter board and the XOIP voice mixing daughter board have the special capability of voice mixing that helps meet some special service demands such as recording both parties' communication voice not in a conference, recording karaoke, etc. For details, refer to the voice mixing function part of the programming guide.

Parameter	Description
ucIsxNo	Node number
ucBrdNo	Board number
ucBrdType	Board type as follows: BT_TRUNK: Trunk daughter board BT_XOIP: XOIP daughter board
pSpecCap	Pointer to the structure address where special capabilities are stored. Refer to the **SPEC_CAP** structure.

Cautions

None.

Errors

None.

Example

```
#include <srllib.h>

UCHAR ucIsxNo     = 0;
UCHAR ucSpanBrd = 0;
UCHAR ucBrdType = BT_TRUNK;
SPEC_CAP spec_cap;
ISX_sr_GetSpecCap(ucIsxNo, ucSpanBrd, ucBrdType, &spec_cap);
printf("Span Mix Cap =%d\n", spec_cap.Span.usMixCap);
```

...

Relevant information

- **ISX_sr_SetMixParm()**
- **ISX_sr_GetMixParm()**

▪ *ISX_sr_SetMixParm()*

Name:	long ISX_sr_SetMixParm(ucIsxNo, ucBrdNo, ucBrdType, pMixParam)	
Input parameters:	**unsigned char ucIsxNo**	Node number
	unsigned char ucBrdNo	Daughter board number
	unsigned char ucBrdType	Daughter board type
	MIX_PARAM *pMixParam	Pointer the address of voice mixing parameters
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_SetMixParm** function sets voice mixing parameters of a specified special daughter board. Daughter boards with the voice mixing capability include the trunk voice mixing daughter board and the XOIP voice mixing daughter board.

Parameter	Description
ucIsxNo	Node number
ucBrdNo	Board number
ucBrdType	Board type as follows: BT_TRUNK: Trunk daughter board BT_XOIP: XOIP daughter board
pMixParam	Pointer the address of voice mixing parameters. Refer to the **MIX_PARAM** structure.

Cautions

None.

Errors

None.

Example

```
#include <srllib.h>

UCHAR ucIsxNo     = 0;
UCHAR ucSpanBrd = 0;
UCHAR ucBrdType = BT_TRUNK;
```

```
MIX_PARAM set_mix_param;
set_mix_param.ucMixType = MIXTYPE_PLAY;
memset(set_mix_param.ucUnmixTimeSlot, 0xff, MAX_SPAN_NUM);
//{{Others are provided with the playback voice mixing function besides time slot 10
        of trunk 1
set_mix_param.ucUnmixTimeSlot[1]=10;
//}}
int iRet=ISX_sr_SetMixParm(ucIsxNo, ucSpanBrd, ucBrdType, &set_mix_param);
if(iRet==-1){
    printf("ISX_sr_SetMixParm    fail. \n");
}
...
```

Relevant information

- **ISX_sr_GetSpecCap()**
- **ISX_sr_GetMixParm()**

▪ *ISX_sr_GetMixParm()*

Name:	long ISX_sr_GetMixParm(ucIsxNo, ucBrdNo, ucBrdType, pMixParam)	
Input parameters:	**unsigned char ucIsxNo**	Node number
	unsigned char ucBrdNo	Daughter board number
	unsigned char ucBrdType	Daughter board type
	MIX_PARAM *pMixParam	Pointer to the address of voice mixing parameters
Return value:	0 success -1 failure	
Header file:	srllib.h voclib.h	
Category:	Utility tool functions	
Mode:	Synchronous	

Description

The **ISX_sr_GetMixParm** function obtains voice mixing parameters of a specified special daughter board. Daughter boards with the voice mixing capability include the trunk voice mixing daughter board and the XOIP voice mixing daughter board.

Parameter	Description
ucIsxNo	Node number
ucBrdNo	Board number
ucBrdType	Board type as follows: BT_TRUNK: Trunk daughter board BT_XOIP: XOIP daughter board
pMixParam	Pointer the address of voice mixing parameters. Refer to the **MIX_PARAM** structure.

Cautions

None.

Errors

None.

Example

```
#include <srllib.h>

UCHAR ucIsxNo     = 0;
UCHAR ucSpanBrd = 0;
UCHAR ucBrdType = BT_TRUNK;
MIX_PARAM get_mix_param;
int iRet=ISX_sr_GetMixParm(ucIsxNo, ucSpanBrd, ucBrdType, &get_mix_param);
if(iRet==-1){
    printf("ISX_sr_GetMixParm    fail. \n");
}
...
```

Relevant information

- **ISX_sr_GetSpecCap()**
- **ISX_sr_SetMixParm()**

2.5 SYS Event List

- ***SYSEV_MC_STATUS***

  See **Event Descriptions**.
- ***SYSEV_PRD_STATUS***

  See **Event Descriptions**.
- ***SYSEV_MB_STATUS***

  See **Event Descriptions**.
- ***SYSEV_DSP_STATUS***

  See **Event Descriptions**.
- ***SYSEV_PRI_STATUS***

  See **Event Descriptions**.
- ***SYSEV_SPAN_STATUS***

  See **Event Descriptions**.
- ***SYSEV_DSP_BRD_CAP***

  See **Event Descriptions**.
- ***SYSEV_PRI_BRD_CAP***

  See **Event Descriptions**.
- ***SYSEV_XOIP_BRD_CAP***

  See **Event Descriptions**.
- ***SYSEV_SPAN_ALARM***

  See **Event Descriptions**.
- ***SYSEV_SPAN_E1***

  See **Event Descriptions**.
- ***SYSEV_SPAN_T1***

  See **Event Descriptions**.
- ***SYSEV_SPAN_J1***

  See **Event Descriptions**.
- ***SYSEV_SS7_STATUS***

  See **Event Descriptions**.
- ***SYSEV_CAP_DATA***

  See **Event Descriptions**.
- ***SYSEV_XOIP_STATUS***

  See **Event Descriptions**.
- ***SYSEV_SS7_BRD_CAP***

  See **Event Descriptions**.
- ***SYSEV_SIP_STATUS***

  See **Event Descriptions**.

- ***SYSEV_SIP_BRD_CAP***

  See **Event Descriptions**.

- ***SYSEV_M3G_STATUS***

  See **Event Descriptions**.

- ***SYSEV_M3G_BRD_CAP***

  See **Event Descriptions**.

- ***SYSEV_M3G_CONN_STATUS***

  See **Event Descriptions**.

- ***Event Descriptions***

  The following table shows the system event types and the event data explanation.

Event Type	Data Length	SYS_EVT_DATA Event Data (stored in the SYS_EVT_DATA structure)	Event Causes, Status Meanings and Handling Suggestions
SYSEV_MC_STATUS	5	pEvtData->u.cStatus (Status)	When MC (master controller) status is changed, the event is generated. The following causes may result in change of MC status: 1: MC service programs are forcedly closed by the user (STATUS_OUT_OF_SRV) or are stopped by the user through OAM (STATUS_REQ_STOP). 2: Network connection between the application and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 3: If the application successfully calls ISX_Api_Init and is successfully connected to a specified MC, the event is generated (STATUS_WORKING). STATUS_RESET: The MC has been reset. All media board devices and channel devices and call channel devices shall be turned off. STATUS_WORKING: The MC is in working state. The SDK can work normally only in this state. STATUS_OUT_OF_SRV: The MC is in out-of-service state. All media and call functions cannot be called in this state. STATUS_REQ_STOP: The user's request is stopped (the reason is that the OAM requests stop or the user presses the "Stop" button of the board). After the event is received, cleanup work shall done as soon as possible.
SYSEV_MB_STATUS	5	pEvtData->cIsxNo (Node number) pEvtData->u.cStatus (Status)	When MB (motherboard) status is changed, the event is generated. The following causes may result in change of MB status: 1: The isx4000 swtich is out of service (STATUS_OUT_OF_SRV) or reset (STATUS_RESET) by the user. 2: Network connection between the isx4000 switch and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 3: If the application successfully calls

			ISX_Api_Init and the MC connected to the APP is successfully connected to the MB, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the motherboard status only. Functions that are affected by the motherboard status include switching, DTI, CS switching channel functions, etc.
SYSEV_DSP_STATUS	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus	When DSP (DSP board) status is changed, the event is generated: The following causes may result in change of DSP board status: 1: The isx4000 switch is out of service or reset by the user, which causes the DSP board to be in STATUS_OUT_OF_SRV state. 2: The DSP board is out of service or restarted by the user (STATUS_OUT_OF_SRV). 3: Network connection between the DSP board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: If the application successfully calls ISX_Api_Init and the MC connected to the APP is successfully connected to the DSP, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the DSP board status only. Functions that are affected by the DSP board status include playback, recording, conferencing and echo cancellation functions, etc.
SYSEV_PRI_STATUS	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus (Status)	When PRI (PRI board) status is changed, the event is generated: The following causes may result in change of PRI board status: 1: The isx4000 switch is out of service or reset by the user, which causes the PRI board to be in STATUS_OUT_OF_SRV state. 2: The PRI board is out of service or restarted by the user (STATUS_OUT_OF_SRV). 3: Network connection between the PRI board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: If the PRI board is hot plugged into the ISX4000 switch, successfully started and connected to the MC, the event is generated (STATUS_WORKING). 5: When the PRI board is hot unplugged from the switch, the event is generated (STATUS_OUT_OF_SRV). 6: If the application successfully calls ISX_Api_Init and the MC connected to the APP is successfully connected to the PRI board, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the PRI board status

			only. Functions that are affected by the PRI board status include Global Call functions.
SYSEV_SS7_STATUS		pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus (Status)	When SS7 (SS7 board) status is changed, the event is generated: The following causes may result in change of SS7 board status: 1: The isx4000 switch is out of service or reset by the user, which causes the SS7 board to be in STATUS_OUT_OF_SRV state. 2: The SS7 board is out of service or restarted by the user (STATUS_OUT_OF_SRV). 3: Network connection between the SS7 board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: The SS7 board is hot plugged from the isx4000 switch (STATUS_OUT_OF_SRV, STATUS_WORKING). 5: If the application successfully calls ISX_Api_Init and the MC connected to the APP is successfully connected to the SS7 board, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the SS7 board status only. Functions that are affected by the SS7 board status include Global Call functions.
SYSEV_XOIP_STATUS		pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus (Status)	When VOIP (VOIP board) status is changed, the event is generated: The following causes may result in change of VOIP board status: 1: The isx4000 switch is out of service or reset by the user, which causes the VOIP board to be in STATUS_OUT_OF_SRV state. 2: The VOIP board is out of service or restarted by the user (STATUS_OUT_OF_SRV). 3: Network connection between the VOIP board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: If the application successfully calls ISX_Api_Init and the MC connected to the APP is successfully connected to the VOIP board, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the XOIP board status only. Functions that are affected by the XOIP board status include ipm functions.
SYSEV_SIP_STATUS		pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus (Status)	When SIP (SIP board) status is changed, the event is generated: The following causes may result in change of SIP board status: 1: The isx4000 switch is out of service or reset by the user, which causes the SIP board to be in STATUS_OUT_OF_SRV state. 2: The SIP board is out of service or restarted by the user (STATUS_OUT_OF_SRV).

			3: Network connection between the SIP board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: The SIP board is hot plugged from the isx4000 switch (STATUS_OUT_OF_SRV, STATUS_WORKING). 5: If the application successfully calls ISX_Api_Init and there are SIP boards that are managed by MC and operate normally, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the SIP board status only. Functions that are affected by the SIP board status include Global Call functions.
SYSEV_M3G_STATUS		pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.cStatus (Status)	When M3G (M3G board) status is changed, the event is generated: The following causes may result in change of M3G board status: 1: The isx4000 switch is out of service or reset by the user, which causes the M3G board to be in STATUS_OUT_OF_SRV state. 2: The M3G board is out of service or restarted by the user (STATUS_OUT_OF_SRV). 3: Network connection between the M3G board and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 4: The M3G board is hot plugged from the isx4000 switch (STATUS_OUT_OF_SRV, STATUS_WORKING). 5: If the application successfully calls ISX_Api_Init and there are M3G boards that are managed by MC and operate normally, the event is generated (STATUS_WORKING). The event is similar to the SYSEV_MC_STATUS event. The difference is that this event describes the M3G board status only. Functions that are affected by the M3G board status include M3G library functions.
SYSEV_M3G_CONN_STATUS		pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.m3gConnSt (Status)	When connection status between M3G (M3G board) and M3GC is changed, the event is generated: The following causes may result in change of M3G board connection status: 1: The isx4000 switch is out of service or reset by the user, which causes the M3G board to be in M3G_BRD_CONN_STATUS_DEACTIVE state. 2: The M3G board is out of service or restarted by the user (M3G_BRD_CONN_STATUS_DEACTIVE). 3: Network connection between the M3G board and the M3GC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV).
SYSEV_SPAN_STATUS	5	pEvtData->cIsxNo (Node	When span (trunk) status is changed, the

		number) pEvtData->cBrdNo (Board number) pEvtData->cSpanNum (Span number) pEvtData->u.cStatus (Status)	event is generated: The following causes may result in change of trunk status: 1: The isx4000 switch is out of service or reset by the user, which causes the trunk to be in STATUS_OUT_OF_SRV state. 2: Correpsonding trunks are stopped bythe user (STATUS_OUT_OF_SRV) 3: The trunks that exceed licence are in STATUS_OUT_OF_SRV state. 4: If the application successfully calls ISX_Api_Init and there are spans that are managed by MC and operate normally, the event is generated (STATUS_WORKING). STATUS_WORKING: Specified spans are in working state. The SDK can work normally only in this state. STATUS_OUT_OF_SRV: Specified spans are in out-of-service state. All media and call functions cannot be called in this state.
SYSEV_DSP_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When DSP board capability is changed, the event is generated: The following causes may result in change of DSP board capability: 1: When the user changes DSP board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are DSP boards that operate normally, the event is generated. DSP daughter board capacity. Reports the number of DSP channels. Only reports the number of DSP channels with license.
SYSEV_PRI_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When PRI board capability is changed, the event is generated: The following causes may result in change of PRI board capability: 1: When the user changes PRI board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are PRI boards that operate normally, the event is generated. PRI daughter board capacity. Reports the number of spans the PRI daughter board serves. Only reports the number of spans with license.
SYSEV_SS7_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When SS7 board capability is changed, the event is generated: The following causes may result in change of SS7 board capability: 1: When the user changes SS7 board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are SS7 boards that operate normally, the event is generated. SS7 daughter board capacity. Reports the number of links the SS7 daughter board

			serves. Only reports the number of links with license.
SYSEV_XOIP_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When VOIP board capability is changed, the event is generated: The following causes may result in change of VOIP board capability: 1: When the user changes VOIP board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are VOIP boards that operate normally, the event is generated. VOIP daughter board capacity. Reports the number of IPM channels. Only reports the number of IPM channels with license.
SYSEV_SIP_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When SIP board capability is changed, the event is generated: The following causes may result in change of SIP board capability: 1: When the user changes SIP board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are SIP boards that operate normally, the event is generated. SIP daughter board capacity. Reports the number of SIP channels. Only reports the number of SIP channels with license.
SYSEV_M3G_BRD_CAP	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->u.sCap (Capacity)	When M3G board capability is changed, the event is generated: The following causes may result in change of M3G board capability: 1: When the user changes M3G board capability through OAM configuration, the event is generated. 2: If the application successfully calls ISX_Api_Init and there are M3G boards that operate normally, the event is generated. M3G daughter board capacity. Reports the number of M3G channels. Only reports the number of M3G channels with license.
SYSEV_SPAN_ALARM	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->cSpanNum (Span number) pEvtData->u.usAlarm (Alarm contents)	When the trunk shows physical abnormalities such as disconnection from phsycal trunk devices, the event is generated. Trunk alarm or alarm recovery Alarm contents are described below.
SYSEV_SPAN_E1	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->cSpanNum (Span number)	When trunk type is changed, the event is generated: The following causes may cause the change of trunk type to E1: 1: The user changes the trunk type from non-E1 type to E1 type through OAM. 2: If the application successfully calls ISX_Api_Init and there are trunks of E1 type, the event is generated.

			Significance: Prompts that the trunk type has been changed to E1
SYSEV_SPAN_T1	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->cSpanNum (Span number)	When trunk type is changed, the event is generated: The following causes may cause the change of trunk type to T1: 1: The user changes the trunk type from non-T1 type to T1 type through OAM. 2: If the application successfully calls ISX_Api_Init and there are trunks of T1 type, the event is generated. Significance: Prompts that the trunk type has been changed to T1
SYSEV_SPAN_J1	5	pEvtData->cIsxNo (Node number) pEvtData->cBrdNo (Board number) pEvtData->cSpanNum (Span number)	When trunk type is changed, the event is generated: The following causes may cause the change of trunk type to J1: 1: The user changes the trunk type from non-J1 type to J1 type through OAM. 2: If the application successfully calls ISX_Api_Init and there are trunks of J1 type, the event is generated. Significance: Prompts that the trunk type has been changed to J1
SYSEV_PRD_STATUS	5	pEvtData->cIsxNo (PRD node number) pEvtData->u.cStatus (Status)	When PRD status is changed, the event is generated: The following causes may result in change of PRD status: 1: PRD service programs are forcedly closed by the user (STATUS_OUT_OF_SRV) or are stopped through OAM (STATUS_OUT_OF_SRV) or activated (STATUS_WORKING). 2: Physical network connection between the PRD service programs and the MC is successful (STATUS_WORKING) or interrupted (STATUS_OUT_OF_SRV). 3: If the application successfully calls ISX_Api_Init and there are PRDs that are managed by the MC and operate normally, STATUS_WORKING is obtained. The event describes PRD status. STATUS_RESET: The PRD has been reset. All opened file handles on the PRD (file handles opened by the ISX_dx_fileopen function) are invalid. STATUS_WORKING: The PRD is in working state. The SDK can call the ISX_dx_fileopen and ISX_dx_fileclose functions only in this state. STATUS_OUT_OF_SRV: The PRD is in out-of-service state. The SDK cannot call the ISX_dx_fileopen and ISX_dx_fileclose functions only in this state.

Trunk alarm contents are two bytes long. Alarm contents are expressed by bit. 0 indicates no alarm. 1 indicates alarm.

Bit0：LOLITC, Loss of Line Interface Transmit Clock

Bit1：TOCD, Transmit Open-Circuit Detector

Bit2：TCLE, Transmit Current Limit Exceeded
Bit3：LRCL, Line Interface Receive Carrier Loss
Bit4：RLOS, Rx Loss Of Sync
Bit5：FRCL, Framer Receive Carrier Loss
Bit6：RUA1, Receive Unframed All Ones
Bit7：RYEL, Receive Yellow Alarm
Bit8-Bit15：Reserved for future use.

2.6 Data Structure

▪ *SPEC_CAP*

```
typedef struct{
UCHAR           ucMixCap;
UCHAR           ucRvrCap[MAX_SPEC_CAP_NUM-1];
}SPAN_CAP_SET;

typedef struct{
UCHAR           ucMixCap;
UCHAR           ucRvrCap[MAX_SPEC_CAP_NUM-1];
}XOIP_CAP_SET;

typedef struct{
union{
    SPEC_CAP_SET Span;
    SPEC_CAP_SET Xoip;
};
}SPEC_CAP;

typedef enum{
    MIXCAP_NON      =0,     //no special capability
    MIXCAP_NORMAL   =1,     //ordinary voice mixing capability
    MIXCAP_HIIMP    =2,     //high-impedance voice mixing capability, valid for only the trunk board
}MIX_CAP;
```

Description

The **SPEC_CAP** data structure stores special capability data of daughter boards. When **ISX_sr_GetSpecCap()** runs, the bottom layer returns the special capability set of daughter boards specified by the user to the buffer specified by the user. The structure of buffer is the **SPEC_CAP** structure.

Field description

Each field of the **SPEC_CAP** data structure is described as follows:

Span

Special capability set for trunk daughter boards.

.ucMixCap: 0 - no special capability；1 - ordinary voice mixing capability；2 - high-impedance voice mixing capability. Refer to MIX_CAP.

Xoip

Special capability set for XOIP daughter boards.

.ucMixCap: 0 - no special capability；1 - ordinary voice mixing capability. Refer to MIX_CAP.

usMixCap

voice mixing capability contained in the capability set.

usRvrCap

Reserved field of the capability set.

Example

For how to use the data structure, refer to the example of **ISX_sr_GetSpecCap()**.

▪ *MIX_PARAM*

```
typedef struct{
    UCHARucMixType;         //voice mixing type
```

```
    UCHARucUnmixTimeSlot[8];    //Trunk time slot number without voice mixing
}MIX_PARAM;
```

Description

The **MIX_PARAM** data structure store voice mixing parameter data of special daughter boards. When **ISX_sr_SetMixParm()** runs, the bottom layer returns the voice mixing parameters of daughter boards specified by the user to the buffer specified by the user. The structure of buffer is the **MIX_PARAM** structure.

Field description

Each field of the **MIX_PARAM** data structure is described as follows:

ucMixType

Voice mixing type. Valid values are specified by the following MIX_TYPE enumeration types.

```
typedef enum{
    MIXTYPE_NULL,       //no voice mixing
    MIXTYPE_RECORD,  //voice mixing for recording
    MIXTYPE_PLAY,       // voice mixing for playback
    MIXTYPE_BOTH,       //voice mixing for both recording and playback
    MIXTYPE_HIIMPREC,//high-impedance voice mixing for recording
}MIX_TYPE;
```

ucUnmixTimeSlot

Trunk time slot number without voice mixing (each element in the array represents a trunk), E1 (0-31), T1/J1 (1-24). 0xFF indicates complete conversion. For XOIP daughter boards with special capabilities, the field is useless.

Example

For how to use the data structure, refer to the example of **ISX_sr_SetMixParm()**.