

# **SCbus Routing Software Reference for Linux**

**Copyright © 2001 Dialogic Corporation**

05-0313-004



## **COPYRIGHT NOTICE**

Copyright © 2001 Dialogic Corporation. All Rights Reserved.

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

All names, products, and services mentioned herein are the trademarks or registered trademarks of their respective organizations and are the sole property of their respective owners. DIALOGIC (including the Dialogic logo), DTI/124, and SpringBoard are registered trademarks of Dialogic Corporation. A detailed trademark listing can be found at: <http://www.dialogic.com/legal.htm>.

Publication Date: September, 2001

Part Number: 05-0313-004

Dialogic, an Intel Company  
1515 Route 10  
Parsippany NJ 07054  
U.S.A.

For **Technical Support**, visit the Dialogic support website at:  
<http://support.dialogic.com>

For **Sales Offices** and other contact information, visit the main Dialogic website at:  
<http://www.dialogic.com>



# Table of Contents

---

<b>1. Introduction .....</b>	<b>1</b>
1.1. SCbus Routing Function Overview .....	1
<b>2. SCbus Convenience Functions.....</b>	<b>5</b>
nr_scroute() - makes a full or half-duplex connection .....	6
nr_scunroute() - breaks a full or half duplex connection .....	12
<b>3. Function Reference.....</b>	<b>17</b>
ag_getctinfo() - returns information about an analog channel device handle.....	18
ag_getxmitslot() - returns SCbus time slot number.....	22
ag_listen() - connects analog listen channel to SCbus time slot.....	26
ag_unlisten() - disconnects analog receive channel from SCbus .....	30
dt_getctinfo() - gets SCbus digital interface information.....	33
dt_getxmitslot() - returns SCbus time slot number .....	37
dt_listen() - connects digital listen channel to SCbus time slot.....	40
dt_unlisten() - disconnects digital receive channel from SCbus .....	44
dx_getctinfo() - returns information about a voice channel device handle .....	47
dx_getxmitslot() - returns SCbus time slot number .....	51
dx_listen() - connects voice listen channel to SCbus time slot .....	54
dx_unlisten() - disconnects voice receive channel from SCbus .....	58
fx_getxmitslot() - returns SCbus time slot number .....	61
fx_listen() - connects fax listen channel to SCbus time slot.....	64
fx_unlisten() - disconnects fax listen channel from SCbus .....	69
<b>4. Data Structure Reference .....</b>	<b>73</b>
4.1. Channel/Time Slot Device Information (CT_DEVINFO).....	73
4.2. SCbus Time Slot Information (SC_TSINFO).....	75
<b>5. Demo Program.....</b>	<b>77</b>
5.1. Answering Machine Demo .....	77
<b>Appendix A - SCbus Routing Function Summary .....</b>	<b>97</b>
<b>Appendix B - Related Publications.....</b>	<b>99</b>
<b>Index .....</b>	<b>101</b>

*SCbus Routing Software Reference for Linux*

# 1. Introduction

---

The SCbus is a real-time, high-speed, time division multiplexed (TDM) communications bus that provides 1024 time slots for transmission of digital information between SCbus products.

This guide describes the SCbus routing functions. These functions provide the ability to establish communications between SCbus devices. An explanation of an SCbus Linux demo program is also provided in this guide. For information on SCbus routing, refer to the *SCbus Routing Guide*.

**NOTE:** The SCbus routing functions only support SCbus hardware configurations.

## 1.1. SCbus Routing Function Overview

The SCbus routing functions comprise network/resource (nr\_) SCbus convenience functions and individual SCbus routing functions. For most routing applications, the following network/resource SCbus convenience functions should suffice. The SCbus convenience functions include all of the functionality of the individual SCbus routing functions.

<b>Nr_scroute()</b>	makes half or full duplex connection between two SCbus devices
<b>nr_scunroute()</b>	breaks half or full duplex connection between two SCbus devices

The network/resource SCbus convenience functions are not part of the voice library. Their C source code is provided in a separate file called *sctools.c* in */usr/dialogic/sctools* directory.

The individual SCbus routing functions provide the ability to program each phase of connecting or disconnecting the receive channel of a device to the transmit channel of another device or to build your own convenience functions. These individual SCbus routing functions can be characterized by their prefix and by their suffix.

The prefix of the individual SCbus routing functions identify the type of device to which the function applies:

<b>ag_</b>	analog device (loop-start interface)
------------	--------------------------------------

**SCbus Routing Software Reference for Linux**

<b>dt_</b>	digital device (T-1 or E-1 digital service interface)
<b>dx_</b>	voice device
<b>fx_</b>	fax device

The suffix of the individual SCbus routing functions identify the operation or task performed by the function:

<b>_getctinfo( )</b>	returns information about a device (analog device, voice device, digital device, or other technology device) in a CT_DEVINFO structure; this function is not used for routing.
<b>_getxmitslot( )</b>	returns the SCbus time slot information into an SC_TSINFO structure that includes the number of the SCbus time slot connected to the transmit channel of the specified device
<b>_listen( )</b>	connects the listen (receive) channel of the specified device to an SCbus time slot
<b>_unlisten( )</b>	disconnects the listen (receive) channel of the specified device from an SCbus time slot

## 1. Introduction

The individual SCbus functions described in this reference are:

<b>ag_getctinfo( )</b>	returns information about an analog device in a CT_DEVINFO structure
<b>ag_getxmitslot( )</b>	returns SCbus time slot information into an SC_TSINFO structure that includes the number of the SCbus time slot connected to the analog transmit channel
<b>ag_listen( )</b>	connects analog receive (listen) channel to an SCbus time slot
<b>ag_unlisten( )</b>	disconnects analog receive (listen) channel from SCbus time slot
<b>dt_getctinfo( )</b>	returns information about a digital interface device in a CT_DEVINFO structure
<b>dt_getxmitslot( )</b>	returns SCbus time slot information into an SC_TSINFO structure that includes the number of the SCbus time slot connected to the digital transmit channel
<b>dt_listen( )</b>	connects digital receive (listen) channel to an SCbus time slot
<b>dt_unlisten( )</b>	disconnects digital receive (listen) channel from SCbus time slot
<b>dx_getctinfo( )</b>	returns information about a voice device in a CT_DEVINFO structure
<b>dx_getxmitslot( )</b>	returns SCbus time slot information into an SC_TSINFO structure that includes the number of the SCbus time slot connected to the voice transmit channel
<b>dx_listen( )</b>	connects receive voice (listen) channel to an SCbus time slot
<b>dx_unlisten( )</b>	disconnects voice receive (listen) channel from SCbus time slot

***SCbus Routing Software Reference for Linux***

<b>fx_getxmitslot()</b>	returns SCbus time slot information into an SC_TSINFO structure that includes the number of the SCbus time slot connected to the fax transmit channel
<b>fx_listen()</b>	connects fax receive (listen) channel to an SCbus time slot
<b>fx_unlisten()</b>	disconnects fax receive (listen) channel from SCbus time slot

## 2. SCbus Convenience Functions

---

The Dialogic *sctools.h* library provides SCbus convenience functions that make the routing process easier for application developers by allowing them to perform a combination of SCbus individual functions using only two function calls. Using the **nr\_scroute()** and **nr\_scunroute()** functions, the application specifies only two device handles and device types, and whether the connection will be full or half-duplex. The SCbus convenience functions are described in this chapter.

**nr\_scroute()**

***makes a full or half-duplex connection***

---

**Name:** int nr\_scroute(devh1,devtype1,devh2,devtype2,mode)  
**Inputs:** int devh1                   • channel device handle for SCbus based board  
          unsigned short devtype1   • type of device for devh1  
          int devh2                   • channel device handle for SCbus based board  
          unsigned short devtype2   • type of device for devh2  
          unsigned char mode       • half or full duplex connection  
**Returns:** 0 on success  
          -1 on error  
**Includes:** stdio.h  
          varargs.h  
          srllib.h  
          dxxlib.h  
          sctools.h  
          Optional: dtilib.h and faxlib.h  
**Category:** Routing convenience  
**Mode:** Synchronous

---

### ■ Description

The **nr\_scroute()** convenience function [makes a full or half-duplex connection](#) between two SCbus time slots connected to devices on SCbus based boards. This convenience function is not a part of any library and is provided in a separate C source file called *sctools.c* in the */user/dialogic/sctools* directory.

The **nr\_sc** prefix to the function signifies network (analog and digital) devices and resource (voice and fax) devices accessible via the SCbus. See the *Digital Network Interface Software Reference for Linux* for digital interface device details and the *Fax Software Reference* for fax device details.

**NOTE:** DTI and fax functionality may be conditionally compiled in or out of the function using the DTISC and FAXSC defines in the makefile provided with the function. To compile in DTI functionality, link with the DTI library. To compile in fax functionality, link with the fax library. Error message printing may also be conditionally compiled in or out by using the PRINTON define in the makefile.

Parameter	Description
<b>devh1:</b>	Specifies the valid channel device handle obtained when the channel was opened for the first device (the transmitting device for half duplex).
<b>devtype1:</b>	Specifies the parameters that characterize the <b>devh1</b> channel device. SC_VOX           voice channel device SC_LSI           analog channel device SC_DTI           digital time slot device SC_FAX           fax channel device
<b>devh2:</b>	Specifies the valid channel device handle obtained when the channel was opened for the second device (the listening device for half duplex).
<b>devtype2:</b>	Specifies the parameters that characterize the <b>devh2</b> channel device. See <b>devtype1</b> for a list of defines.
<b>mode:</b>	Specifies half or full duplex connection. This parameter contains one of the following defines from <i>sctools.h</i> to specify full or half duplex: SC_FULLDUP    full duplex connection SC_HALFDUP    half duplex connection. The default mode value is SC_FULLDUP. When SC_HALFDUP is specified, then the function returns with the second device listening to the SCbus time slot connected to the first device.

■ **Cautions**

- The **devtype1** and **devtype2** parameters must match the types of the device handles in **devh1** and **devh2**.
- If you have not defined DTISC and FAXSC when compiling the *sctools.c* file, you cannot use this function to route digital channels or fax channels.
- If you have not defined PRINTON in the makefile, errors will not be displayed.

## ■ Source Code

```

#include <stdio.h>
#include <varargs.h>
#include <srllib.h>
#include <dxlib.h>
#ifdef DTISC
#include <dtilib.h>
#endif
#ifdef FAXSC
#include <faxlib.h>
#endif

#include "sctools.h"

#if ( defined( __STDC__ ) || defined( __cplusplus ) )
int nr_scroute( int devh1, unsigned short devtype1,
               int devh2, unsigned short devtype2, unsigned char mode )
#else
int nr_scroute( devh1, devtype1, devh2, devtype2, mode )
    int devh1;
    unsigned short devtype1;
    int devh2;
    unsigned short devtype2;
    unsigned char mode;
#endif
{
    SC_TSINFO sc_tsinfo; /* SCbus time slot info structure */
    long scts; /* SCbus time slot */

    /*
     * Setup the SCbus time slot information structure.
     */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarray = &scts;
    /*
     * Get the SCbus time slot connected to the transmit of the first
     * device.
     */
    switch (devtype1) {
    case SC_VOX:
        if (dx_getxmitslot(devh1, &sc_tsinfo) == -1) {
            nr_scerrror("nr_scroute: %s: dx_getxmitslot ERROR: %s\n",
                       ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
            return -1;
        }
        break;

    case SC_LSI:
        if (ag_getxmitslot(devh1, &sc_tsinfo) == -1) {
            nr_scerrror("nr_scroute: %s: ag_getxmitslot ERROR: %s\n",
                       ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
            return -1;
        }
        break;
#ifdef DTISC
    case SC_DTI:
        if (dt_getxmitslot(devh1, &sc_tsinfo) == -1) {
            nr_scerrror("nr_scroute: %s: dt_getxmitslot ERROR: %s\n",
                       ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
            return -1;
        }
        break;

```

```

#endif

#ifdef FAXSC
case SC_FAX:
    if (fx_getxmitslot(devh1, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: fx_getxmitslot ERROR: %s\n",
                    ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;
#endif

default:
    nr_scerrror("nr_scroute: %s: ERROR: Invalid 1st device type\n",
                ATDV_NAMEP(devh1));
    return -1;
}

/*
 * Make the second device type listen to the time slot that the first
 * device is transmitting on. If a half duplex connection is desired,
 * then return. Otherwise, get the SCbus time slot connected to the
 * transmit of the second device.
 */
switch (devtype2) {
case SC_VOX:
    if (dx_listen(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: dx_listen ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    if (mode == SC_HALFDUP) {
        return 0;
    }
    if (dx_getxmitslot(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: dx_getxmitslot ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    break;

case SC_LSI:
    if (ag_listen(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: ag_listen ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    if (mode == SC_HALFDUP) {
        return 0;
    }
    if (ag_getxmitslot(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: ag_getxmitslot ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    break;

#ifdef DTISC
case SC_DTI:
    if (dt_listen(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: dt_listen ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    if (mode == SC_HALFDUP) {

```

**nr\_scroute()***makes a full or half-duplex connection*

```
    return 0;
}

if (dt_getxmitslot(devh2, &sc_tsinfo) == -1) {
    nr_scerrror("nr_scroute: %s: dt_getxmitslot ERROR: %s\n",
                ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
    return -1;
}

break;
#endif

#ifdef FAXSC
case SC_FAX:
    if (fx_listen(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: fx_listen ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    if (mode == SC_HALFDUP) {
        return 0;
    }
    if (fx_getxmitslot(devh2, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: fx_getxmitslot ERROR: %s\n",
                    ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
        return -1;
    }
    break;
#endif

default:
    nr_scerrror("nr_scroute: %s: ERROR: Invalid 2nd device type\n",
                ATDV_NAMEP(devh2));
    return -1;
}
/*
 * Now make the first device listen to the transmit Sbus time slot
 * of the second device.
 */
switch (devtype1) {
case SC_VOX:
    if (dx_listen(devh1, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: dx_listen ERROR: %s\n",
                    ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;

case SC_LSI:
    if (ag_listen(devh1, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: ag_listen ERROR: %s\n",
                    ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;

#ifdef DTISC
case SC_DTI:
    if (dt_listen(devh1, &sc_tsinfo) == -1) {
        nr_scerrror("nr_scroute: %s: dt_listen ERROR: %s\n",
                    ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;
#endif
}
#endif
```

```
#ifndef FAXSC
case SC_FAX:
    if (fx_listen(devhl, &sc_tsinfo) == -1) {
        nr_scerror("nr_scroue: %s: fx_listen ERROR: %s\n",
                  ATDV_NAMEP(devhl), ATDV_ERRMSGP(devhl));
        return -1;
    }
    break;
#endif

    }
    return 0;
}

static void nr_scerror(va_alist)
    va_dcl
{
#ifdef PRINTON
    va_list args;
    char *fmt;

    /*
     * Make args point to the 1st unnamed argument and then print
     * to stderr.
     */
    va_start(args);
    fmt = va_arg(args, char *);
    vfprintf(stderr, fmt, args);
    va_end(args);
#endif
}
```

■ **See Also**

- **nr\_scunroute()**

---

**nr\_scunroute( )****breaks a full or half duplex connection**

---

**Name:** int nr\_scunroute(devh1, devtype1, devh2, devtype2, mode)  
**Inputs:** int devh1                   • channel device handle for SCbus based board  
          unsigned short devtype1   • type of device for devh1  
          int devh2                   • channel device handle for SCbus based board  
          unsigned short devtype2   • type of device for devh2  
          unsigned char mode         • half or full duplex connection  
**Returns:** 0 on success  
          -1 on error  
**Includes:** stdio.h  
          varargs.h  
          srlib.h  
          dxxlib.h  
          sctools.h  
          Optional: dtilib.h and faxlib.h  
**Category:** Routing convenience  
**Mode:** Synchronous

---

**■ Description**

The **nr\_scunroute( )** convenience function [breaks a full or half duplex connection](#) between two SCbus time slots connected to resources on SCbus based boards.

This function is not a part of any library and is provided in a separate C source file called *sctools.c* in the */user/dialogic/sctools* directory.

The **nr\_sc** prefix to the function signifies network (analog and digital) devices and resource (voice and fax) devices accessible via the SCbus. See the *Digital Network Interface Software Reference for Linux* for digital interface device details and the *Fax Software Reference for Linux* for fax device details.

**NOTE:** DTI or fax functionality may be conditionally compiled in or out of the function using the DTISC and/or FAXSC defines in the makefile provided with the function. To compile in DTI functionality, link with the DTI library. To compile in fax functionality, link with the fax library. Error message printing may also be conditionally compiled in or out by using the PRINTON define in the makefile.

Parameter	Description
<b>devh1:</b>	Specifies the valid channel device handle obtained when the channel was opened for the first device.
<b>devtype1:</b>	Specifies the parameters that characterize the <b>devh1</b> channel device. SC_VOX      voice channel device SC_LSI      analog channel device SC_DTI      digital time slot device SC_FAX      fax channel device
<b>devh2:</b>	Specifies the valid channel device handle obtained when the channel was opened for the second device
<b>devtype2:</b>	Specifies the parameters that characterize the <b>devh2</b> channel device. See <b>devtype1</b> for a list of defines.
<b>mode:</b>	Specifies half or full duplex connection. This parameter contains one of the following defines from <i>sctools.h</i> to specify full or half duplex: SC_FULLDUP    full duplex connection SC_HALFDUP    half duplex connection. The default mode value is SC_FULLDUP. When SC_HALFDUP is specified, then the function returns with the second device NOT listening (receive disconnected) to the SCbus time slot connected to the first device.

#### ■ Cautions

- The **devtype1** and **devtype2** parameters must match the types of the device handles in **devh1** and **devh2**.
- If you have not defined DTISC and FAXSC when compiling the *sctools.c* file, you cannot use this function to route digital channels or fax channels.
- If you have not defined PRINTON in the makefile, errors will not be displayed.

## ■ Source Code

```

#include <stdio.h>
#include <varargs.h>
#include <srllib.h>
#include <dxlib.h>
#ifdef DTISC
#include <dtilib.h>
#endif
#ifdef FAXSC
#include <faxlib.h>
#endif

#include "sctools.h"

#if ( defined( __STDC__ ) || defined( __cplusplus ) )
int nr_scunroute( int devh1, unsigned short devtype1,
                 int devh2, unsigned short devtype2, unsigned char mode )
#else
int nr_scunroute( devh1, devtype1, devh2, devtype2, mode )
    int devh1;
    unsigned short devtype1;
    int devh2;
    unsigned short devtype2;
    unsigned char mode;
#endif
{
    /*
     * Disconnect listen of second device from SCbus time slot-.
     */
    switch (devtype2) {
    case SC_VOX:
        if (dx_unlisten(devh2) == -1) {
            nr_scerror("nr_scunroute: %s: dx_unlisten ERROR: %s\n",
                      ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
            return -1;
        }
        break;

    case SC_LSI:
        if (ag_unlisten(devh2) == -1) {
            nr_scerror("nr_scunroute: %s: ag_unlisten ERROR: %s\n",
                      ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
            return -1;
        }
        break;

#ifdef DTISC
    case SC_DTI:
        if (dt_unlisten(devh2) == -1) {
            nr_scerror("nr_scunroute: %s: dt_unlisten ERROR: %s\n",
                      ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
            return -1;
        }
        break;
#endif

#ifdef FAXSC
    case SC_FAX:
        if (fx_unlisten(devh2) == -1) {
            nr_scerror("nr_scunroute: %s: fx_unlisten ERROR: %s\n",
                      ATDV_NAMEP(devh2), ATDV_ERRMSGP(devh2));
            return -1;
        }
        break;
#endif
    }
}

```

```

    }
    break;
#endif

default:
    nr_scerror("nr_scunroute: %s: ERROR: Invalid 2nd device type\n",
              ATDV_NAMEP(devh2));
    return -1;
}
/*
 * A half duplex connection has already been broken.  If this is all
 * that is required, then return now.
 */
if (mode == SC_HALFDUP) {
    return 0;
}
/*
 * Disconnect listen of first device from SBus time slot
 */
switch (devtypel) {
case SC_VOX:
    if (dx_unlisten(devh1) == -1) {
        nr_scerror("nr_scunroute: %s: dx_unlisten ERROR: %s\n",
                  ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;

case SC_LSI:
    if (ag_unlisten(devh1) == -1) {
        nr_scerror("nr_scunroute: %s: ag_unlisten ERROR: %s\n",
                  ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;

#ifdef DTISC
case SC_DTI:
    if (dt_unlisten(devh1) == -1) {
        nr_scerror("nr_scunroute: %s: dt_unlisten ERROR: %s\n",
                  ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;
#endif

#ifdef FAXSC
case SC_FAX:
    if (fx_unlisten(devh1) == -1) {
        nr_scerror("nr_scunroute: %s: fx_unlisten ERROR: %s\n",
                  ATDV_NAMEP(devh1), ATDV_ERRMSGP(devh1));
        return -1;
    }
    break;
#endif

default:
    nr_scerror("nr_scunroute: %s: ERROR: Invalid 1st device type\n",
              ATDV_NAMEP(devh1));
    return -1;
}
return 0;
}

```

**nr\_scunroute( )*****breaks a full or half duplex connection***

---

```
static void nr_scerror(va_alist)
                    va_dcl
{
#ifdef PRINTON
    va_list args;
    char    *fmt;
    /*
     * Make args point to the 1st unnamed argument and then print
     * to stderr.
     */
    va_start(args);
    fmt = va_arg(args, char *);
    fprintf(stderr, fmt, args);
    va_end(args);
#endif
}
```

**■ See Also**

- **nr\_scroute( )**

## **3. Function Reference**

---

A detailed reference of the library functions used for connecting the receive channel of a device to a transmitting channel of another device via the SCbus is provided in this chapter.

---

***ag\_getctinfo()*** *returns information about an analog channel device handle*

---

**Name:** int ag\_getctinfo(chdev, ct\_devinfo)  
**Inputs:** int chdev                   • analog channel device handle  
          CT\_DEVINFO             • pointer to device information  
          \*ct\_devinfo            structure  
**Returns:** 0 on success  
          -1 on error  
**Includes:** dxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **ag\_getctinfo()** function [returns information about an analog channel device handle](#). This function provides information about the channel device handle of an analog device such as a D/41ESC or D/160SC-LS.

<b>Parameter</b>	<b>Description</b>
<b>chdev:</b>	Specifies the valid analog channel device handle obtained when the channel was opened using <b>dx_open()</b> .
<b>ct_devinfo:</b>	Specifies a pointer to the data structure CT_DEVINFO.

On return from the function, the CT\_DEVINFO structure contains the relevant information and is declared as follows:

```
typedef struct {  
    unsigned long ct_prodid;  
    unsigned char ct_devfamily;  
    unsigned char ct_devmode;  
    unsigned char ct_nettype;  
    unsigned char ct_busmode;  
    unsigned char ct_busencoding;  
    unsigned char ct_rfu[7];  
} CT_DEVINFO;
```

**returns information about an analog channel device handle `ag_getctinfo()`**

Valid values for each member of the CT\_DEVINFO structure are defined in *dxxlib.h*.

<b>Field</b>	<b>Description</b>
<b>ct_prodid</b>	Contains a valid Dialogic product identification number for the device
<b>ct_devfamily</b>	Specifies the device family and may contain either: CT_DFD41E            analog channel of a D/xxE board such as D/41ESC CT_DFSPAN            analog channel of a DIALOG/HD high density board such as D/160SC-LS
<b>ct_devmode</b>	Specifies a device mode field that is valid only for the D/xxE board and may contain one of the following: CT_DMRESOURCE    analog channel not in use CT_DMNETWORK    analog channel available to process calls from the telephone network
<b>ct_nettype</b>	Specifies the type of network interface for the device. Valid values are: CT_NTNONE            D/xxE board configured as a resource device; voice channels available for call processing; analog channels are disabled. CT_NTANALOG        analog and voice devices on board are handling call processing
<b>ct_busmode</b>	Specifies the bus architecture used to communicate with other devices in the system CT_BMSCBUS            SCbus architecture
<b>ct_busencoding</b>	Describes the PCM encoding used on the bus. Valid values are: CT_BEULAW            Mu-law encoding CT_BEALAW            A-law encoding

■ **Cautions**

This function will fail if an invalid channel device handle is specified.

## ***ag\_getctinfo()*** returns information about an analog channel device handle

---

### ■ Example

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

main()
{
    int chdev; /* Channel device handle */
    CT_DEVINFO ct_devinfo; /* Device information structure */

    /* Open board 1 channel 1 devices */
    if ((chdev = dx_open("dxxxBlC1", 0)) == -1) {
        printf("Cannot open channel dxxxBlC1.  errno = %d", errno);
        exit(1);
    }

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

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

*returns information about an analog channel device handle* `ag_getctinfo()`

---

### ■ Errors

If the function returns -1, use the SRL Standard Attribute function `ATDV_LASTERR()` to obtain the error code or use `ATDV_ERRMSGP()` to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
<code>EDX_BADPARAM</code>	Parameter error
<code>EDX_SH_BADEXTTS</code>	SCbus time slot is not supported at current clock rate
<code>EDX_SH_BADINDX</code>	Invalid Switch Handler library index number
<code>EDX_SH_BADTYPE</code>	Invalid channel type (voice, analog, etc.)
<code>EDX_SH_CMDBLOCK</code>	Blocking command is in progress
<code>EDX_SH_LIBBSY</code>	Switch Handler library busy
<code>EDX_SH_LIBNOTINIT</code>	Switch Handler library uninitialized
<code>EDX_SH_MISSING</code>	Switch Handler is not present
<code>EDX_SH_NOCLK</code>	Switch Handler clock fallback failed
<code>EDX_SYSTEM</code>	Linux system error

### ■ See Also

- `dt_getctinfo()`
- `dx_getctinfo()`



*returns SCbus time slot number*

*ag\_getxmitslot( )*

---

The SC\_TSINFO structure is declared as follows:

```
typedef struct {
    unsigned long sc_numts;
    long *sc_tsarray;
} SC_TSINFO;
```

The **sc\_numts** member of the SC\_TSINFO structure must be initialized with the number of SCbus time slots requested (1 for an analog channel). The **sc\_tsarray** member of the SC\_TSINFO structure must be initialized with a pointer to a valid array. Upon return from the function, the array will contain the number (between 0 and 1023) of the SCbus time slot on which the analog channel transmits. An analog channel can transmit on only one SCbus time slot.

#### ■ Cautions

This function will fail when an invalid channel device handle is specified.

***ag\_getxmitslot()***

***returns SCbus time slot number***

---

■ **Example**

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

main()
{
    int chdev;          /* Channel device handle */
    SC_TSINFO sc_tsinfo; /* Time slot information structure */
    long scts;         /* SCbus time slot */

    /* Open board 1 channel 1 devices */
    if ((chdev = dx_open("dxxxB1C1", 0)) == -1) {
        printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
        exit(1);
    }

    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarray = &scts;

    /* Get SCbus time slot connected to transmit of analog channel 1 on board ...1 */
    if (ag_getxmitslot(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }

    printf("%s is transmitting on SCbus time slot %d", ATDV_NAMEP(chdev), ...scts);
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function

**ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADINDX	Invalid Switch Handler library index number
EDX_SH_BADLCTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.) number
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel is already disconnected from SCbus time slot
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux system error

**■ See Also**

- **dt\_listen()**
- **dx\_listen()**
- **fx\_listen()**

***ag\_listen()*** ***connects analog listen channel to SCbus time slot***

---

**Name:** int ag\_listen (chdev, sc\_tsinfo)  
**Inputs:** int chdev • analog channel device handle  
SC\_TSINFO \*sc\_tsinfo • pointer to SCbus time slot information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** dxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **ag\_listen()** function **connects analog listen channel to SCbus time slot**. This function uses the information stored in the SC\_TSINFO structure to connect the analog receive (listen) channel such as on a D/41ESC or D/160SC-LS to an SCbus time slot. This function sets up a half-duplex connection. For a full-duplex connection, the receive (listen) channel of the other device must be connected to the analog transmit channel.

Due to analog signaling processing on voice boards with on-board analog devices, a voice device and its corresponding analog device (analog device 1 to voice device 1, etc.) comprise a single channel. At system initialization, default SCbus routing is to connect these devices in full-duplex communications. See the *SCbus Routing Software Reference for Unix* for more information.

**NOTE:** SCbus convenience function **nr\_scroute()** includes **ag\_listen()** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

Parameter	Description
<b>chdev:</b>	Specifies the valid analog channel device handle obtained when the channel was opened using <b>dx_open()</b> .
<b>sc_tsinfo:</b>	Specifies a pointer to the data structure SC_TSINFO.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {
    unsigned long sc_numts;
    long *sc_tsarray;
} SC_TSINFO;
```

The **sc\_numts** member of the SC\_TSINFO structure must be initialized with the number 1. The **sc\_tsarray** member of the SC\_TSINFO structure must be initialized with a pointer to a valid array that contains a valid SCbus time slot number. Upon return from the function, the listen of the analog channel will be connected to this SCbus time slot.

The SCbus time slot number contained in the array pointed to by **sc\_tsarray** must be obtained, prior to calling **ag\_listen( )**, by calling the appropriate **xx\_getxmitslot( )** function (xx = ag, dt, dx or fx).

Although multiple analog channels may listen (be connected) to the same SCbus time slot, the analog receive (listen) channel can connect to only one SCbus time slot.

#### ■ Cautions

This function will fail when:

- An invalid channel device handle is specified.
- An invalid SCbus time slot number is specified.

**■ Example**

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

main()
{
    int chdev;                /* Channel device handle */
    SC_TSINFO sc_tsinfo;     /* Time slot information structure */
    long scts;               /* SCbus time slot */

    /* Open board 1 channel 1 devices */
    if ((chdev = dx_open("dxxxB1C1", 0)) == -1) {
        printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
        exit(1);
    }

    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tarrayp = &scts;

    /* Get SCbus time slot connected to transmit of voice channel 1 on board 1 */
    if (dx_getxmitslot(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }

    /* Connect the receive of analog channel 1 on board 1 to SCbus
       time slot of voice channel 1 */
    if (ag_listen(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	SCbus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel local time slot type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLTSCNCT	Channel is already connected to SCbus time slot
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux system error

**■ See Also**

- **dx\_getxmitslot()**
- **dt\_getxmitslot()**
- **fx\_getxmitslot()**
- **ag\_unlisten()**

---

***ag\_unlisten( )***                      ***disconnects analog receive channel from SCbus***

---

**Name:** int ag\_unlisten(chdev)  
**Inputs:** int chdev                      • analog channel device handle  
**Returns:** 0 on success  
              -1 on error  
**Includes:** dxxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **ag\_unlisten( )** function **disconnects analog receive channel from SCbus**. This function disconnects the analog receive (listen) channel such as on a D/41ESC or D/160SC-LS from the SCbus time slot.

Calling the **ag\_listen( )** function to connect to a different SCbus time slot will automatically break an existing connection. Thus, when changing connections, you need not call the **ag\_unlisten( )** function.

**NOTE:** SCbus convenience function **nr\_scunroute( )** includes **ag\_unlisten( )** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

<b>Parameter</b>	<b>Description</b>
<b>chdev:</b>	Specifies the valid analog channel device handle obtained when the channel was opened using <b>dx_open( )</b> .

■ **Cautions**

This function will fail when an invalid channel device handle is specified.

■ **Example**

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

main()
{
    int chdev;          /* Voice Channel handle */

    /* Open board 1 channel 1 device */
    if ((chdev = dx_open("dxxxBlC1", 0)) == -1) {
        printf("Cannot open channel dxxxBlC1.  errno = %d", errno);
        exit(1);
    }

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

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel local time slot type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel already disconnected from SCbus time slot
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux system error

**■ See Also**

- **ag\_listen()**

---

**Name:** int dt\_getctinfo(devh, ct\_devinfofop)  
**Inputs:** int devh                      • digital network interface time slot device handle  
    CT\_DEVINFO                      • pointer to device information structure  
    \*ct\_devinfofop  
**Returns:** 0 on success  
    -1 on error  
**Includes:** srllib.h  
    dtilib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The [dt\\_getctinfo\( \)](#) function [gets SCbus digital interface information](#). This function returns information about the digital network interface device associated with the specified digital channel (time slot) (dtiBxTx) such as on a D/240SC-T1 or D/300SC-E1.

Parameter	Description
<b>devh:</b>	Specifies the valid digital network interface time slot device handle returned by a call to <a href="#">dt_open( )</a> .
<b>ct_devinfofop:</b>	Specifies the pointer to the data structure CT_DEVINFO.

On return from the function, the CT\_DEVINFO structure contains the relevant information and is declared as follows:

```
typedef struct {
    unsigned long ct_prodid;
    unsigned char ct_devfamily;
    unsigned char ct_devmode;
    unsigned char ct_nettype;
    unsigned char ct_busmode;
    unsigned char ct_busencoding;
    unsigned char ct_rfu[7];
} CT_DEVINFO;
```

Valid values for each member of the CT\_DEVINFO structure are defined in [dtilib.h](#).

Field	Description
<b>ct_prodid</b>	Contains a valid Dialogic product identification number for the device



**■ Example**

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

main( )
{
    int devh;                /* Digital interface device handle */
    CT_DEVINFO ct_devinfo;  /* Device information structure */

    * Open board 1 time slot 1 on digital interface device */
    if ((devh = dt_open("dtiB1T1", 0)) == -1) {
        printf("Cannot open time slot dtiB1T1.  errno = %d", errno);
        exit(1);
    }

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

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

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDT_BADBRDERR	Board missing or defective
EDT_BADCMDERR	Invalid command parameter to driver
EDT_FWERR	Firmware returned an error
EDT_INVTS	Invalid time slot device handle
EDT_INVMSG	Invalid message
EDT_SH_BADLCLTS	Invalid local time slot number
EDT_SH_BADINDX	Invalid Switch Handler library index number
EDT_SH_BADTYPE	Invalid local time slot type
EDT_SH_LIBBSY	Switch Handler library busy
EDT_SH_LIBNOTINIT	Switch Handler library is uninitialized
EDT_SH_MISSING	Switch Handler is not present
EDT_SH_NOCLK	Switch Handler clock fallback failed
EDT_SYSTEM	Linux system error
EDT_TMOERR	Timed out waiting for reply from firmware

**■ See Also**

- **ag\_getctinfo()**
- **dx\_getctinfo()**

returns SCbus time slot number

**dt\_getxmitslot()**

---

**Name:** int dt\_getxmitslot (devh, sc\_tsinfof)  
**Inputs:** int devh • digital network interface device time slot  
SC\_TSINFO \*sc\_tsinfof • pointer to SCbus time slot information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** srllib.h  
dtlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The **dt\_getxmitslot()** function returns SCbus time slot number of the digital transmit channel. The SCbus time slot information is contained in an SC\_TSINFO structure that includes the number of the SCbus time slot connected to the digital network interface device transmit channel (T-1/E-1 time slot) such as on a D/240SC-T1 or D/300SC-E1 board.

**NOTE:** SCbus convenience function **nr\_scroute()** includes **dt\_getxmitslot()** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

Parameter	Description
<b>devh:</b>	Specifies a digital interface time slot device handle returned by a call to <b>dt_open()</b> .
<b>sc_tsinfof:</b>	Points to the device information structure SC_TSINFO.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {
    unsigned long sc_numts;
    long *sc_tsarray;
} SC_TSINFO;
```

The **sc\_numts** field of the SC\_TSINFO structure must be initialized with the number of SCbus time slots requested (1 for a digital network interface device time slot). The **sc\_tsarray** member of the SC\_TSINFO structure must be initialized with a pointer to a valid array. Upon return from the function, the array will contain the number (between 0 and 1023) of the SCbus time slot on which the digital network interface device transmits.

A digital network interface device can transmit on only one SCbus time slot.

**■ Cautions**

This function will fail when an invalid digital channel (T-1/E-1 time slot/device handle) is specified.

**■ Example**

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

main( )
{
    int devh; /* Time slot device handle */
    SC_TSINFO sc_tsinfo; /* Time slot information structure */
    long scts; /* SCbus time slot */

    /* Open board 1 time slot 1 for digital interface device */
    if ((devh = dt_open("dtiB1T1", 0)) == -1) {
        printf("Cannot open time slot dtiB1T1.  errno = %d", errno);
        exit(1);
    }

    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarrayp = &scts;

    /* Get SCbus time slot connected to transmit of time slot (digital channel) 1 on
board 1 */
    if (dt_getxmitslot(devh, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(devh));
        exit(1);
    }

    printf("%s is transmitting on SCbus time slot %d", ATDV_NAMEP(devh), scts);
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDT_BADBRDERR	Board missing or defective
EDT_BADCMDERR	Invalid command parameter to driver
EDT_FWERR	Firmware returned an error
EDT_INVTS	Invalid time slot device handle
EDT_INVMSG	Invalid message
EDT_SH_BADLCLTS	Invalid local time slot number
EDT_SH_BADINDX	Invalid Switch Handler library index number

*returns SCbus time slot number*

*dt\_getxmitslot()*

---

<b>Equate</b>	<b>Returned When</b>
EDT_SH_BADMODE	Invalid Switch Handler bus configuration
EDT_SH_BADTYPE	Invalid local time slot type
EDT_SH_LCLDSCNCT	Local time slot is already disconnected from SCbus
EDT_SH_LIBBSY	Switch Handler library busy
EDT_SH_LIBNOTINIT	Switch Handler library is uninitialized
EDT_SH_MISSING	Switch Handler is not present
EDT_SH_NOCLK	Switch Handler clock fallback failed
EDT_SYSTEM	Linux system error
EDT_TMOERR	Timed out waiting for reply from firmware

■ **See Also**

- **ag\_listen()**
- **fx\_listen()**
- **dx\_listen()**

---

***dt\_listen()*** ***connects digital listen channel to SCbus time slot***

---

**Name:** int dt\_listen (devh, sc\_tsinfop)  
**Inputs:** int devh • digital interface device handle  
SC\_TSINFO \*sc\_tsinfop • pointer to SCbus time slot information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** srllib.h  
dtlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **dt\_listen()** function [connects digital listen channel to SCbus time slot](#). This function uses the information stored in the SC\_TSINFO structure to connect the digital receive (listen) channel (T-1/E-1 time slot) such as on a D/240SC-T1 or D/300SC-E1 board to an SCbus time slot. This function sets up a half-duplex connection. For a full-duplex connection, the receive (listen) channel of the other device must be connected to the digital transmit channel.

**NOTE:** SCbus convenience function **nr\_scroute()** includes **dt\_listen()** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

<b>Parameter</b>	<b>Description</b>
<b>devh:</b>	Specifies a digital network interface time slot device handle returned by a call to <b>dt_open()</b> .
<b>sc_tsinfop:</b>	Specifies the pointer to the SC_TSINFO data structure.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {  
    unsigned long sc_numts;  
    long *sc_tsarray;  
} SC_TSINFO;
```

The **sc\_numts** field of the SC\_TSINFO structure must be set to 1. The **sc\_tsarray** field of the SC\_TSINFO structure must be initialized with a pointer to a valid array. The first element of this array must contain a valid SCbus time slot number (between 0 and 1023) which was obtained by issuing an **xx\_getxmitslot()** function (xx = ag, dt, dx or fx). Upon return from the **dt\_listen()** function, the digital receive channel will be connected to this time slot.

Although multiple SCbus device channels may listen (be connected) to the same SCbus time slot, a digital receive (listen) channel can connect to only one SCbus time slot.

#### ■ Cautions

This function will fail when:

- An invalid device handle is specified.
- An invalid SCbus time slot number is specified.

**■ Example**

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

main( )
{
    int voxh;                /* Voice channel device handle */
    int dtih;                /* Digital channel (time slot) device handle */
    SC_TSINFO sc_tsinfo;    /* Time slot information structure */
    long scts;              /* SCbus time slot */

    /* Open board 1 channel 1 device */
    if ((voxh = dx_open("dxxxBlC1", 0)) == -1) {
        printf("Cannot open channel dxxxBlC1. errno = %d", errno);
        exit(1);
    }

    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tarrayp = &scts;

    /* Get SCbus time slot connected to transmit of voice channel 1 on board 1 */
    if (dx_getxmitslot(voxh, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(voxh));
        exit(1);
    }

    /* Open board 1 time slot 1 on digital interface device */
    if ((dtih = dt_open("dtiBlT1", 0)) == -1) {
        printf("Cannot open time slot dtiBlT1. errno = %d", errno);
        exit(1);
    }

    /* Connect the receive of digital channel (time slot) 1 on board 1 to SCbus transmit
    time slot of voice channel 1*/
    if (dt_listen(dtih, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(dtih));
        exit(1);
    }
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDT_BADBRDERR	Board missing or defective
EDT_BADCMDERR	Invalid command parameter to driver
EDT_FWERR	Firmware returned an error
EDT_INVTS	Invalid time slot device handle
EDT_INVMSG	Invalid message
EDT_SH_BADLCLTS	Invalid local time slot number
EDT_SH_BADEXTTS	External time slot unsupported at current clock rate
EDT_SH_BADINDX	Invalid Switch Handler library index number
EDT_SH_BADMODE	Invalid Switch Handler bus configuration
EDT_SH_BADTYPE	Invalid local time slot type
EDT_SH_LCLTSCNCT	Local time slot is already connected to SCbus
EDT_SH_LIBBSY	Switch Handler library busy
EDT_SH_LIBNOTINIT	Switch Handler library is uninitialized
EDT_SH_MISSING	Switch Handler is not present
EDT_SH_NOCLK	Switch Handler clock fallback failed
EDT_SYSTEM	Linux system error
EDT_TMOERR	Timed out waiting for reply from firmware

**■ See Also**

- **dt\_unlisten()**
- **ag\_getxmitslot()**
- **dx\_getxmitslot()**
- **fx\_getxmitslot()**



**■ Example**

```

#include <srllib.h>
#include <dtilib.h>
#include <errno.h>

main( )
{
    int devh;                /* Digital channel (time slot) device handle */

    /* Open board 1 time slot 1 device */
    if ((devh = dt_open("dtiB1T1", 0)) == -1) {
        printf("Cannot open time slot dtiB1T1.  errno = %d", errno);
        exit(1);
    }

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

```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDT_BADBRDERR	Board missing or defective
EDT_BADCMDERR	Invalid command parameter to driver
EDT_FWERR	Firmware returned an error
EDT_INVTS	Invalid time slot device handle
EDT_INVMSG	Invalid message
EDT_SH_BADLCLTS	Invalid local time slot number
EDT_SH_BADEXTTS	External time slot unsupported at current clock rate
EDT_SH_BADINDX	Invalid Switch Handler library index number
EDT_SH_BADMODE	Invalid Switch Handler bus configuration
EDT_SH_BADTYPE	Invalid local time slot type
EDT_SH_LCLDSCNCT	Local time slot is already disconnected from SCbus
EDT_SH_LIBBSY	Switch Handler library busy
EDT_SH_LIBNOTINIT	Switch Handler library is uninitialized

***dt\_unlisten()***

***disconnects digital receive channel from SCbus***

---

**Equate**

EDT\_SH\_MISSING

EDT\_SH\_NOCLK

EDT\_SYSTEM

EDT\_TMOERR

**Returned When**

Switch Handler is not present

Switch Handler clock fallback failed

Linux system error

Timed out waiting for reply from firmware

■ **See Also**

- **dt\_listen()**

---

**returns information about a voice channel device handle**      **dx\_getctinfo( )**

---

**Name:** int dx\_getctinfo(chdev, ct\_devinfo)  
**Inputs:** int chdev                      • channel device handle  
          CT\_DEVINFO                    • pointer to device information  
          \*ct\_devinfo                    structure  
**Returns:** 0 on success  
          -1 on error  
**Includes:** dxxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **dx\_getctinfo( )** function returns information about a voice channel device handle. This function provides information about a voice channel device handle.

Parameter	Description
<b>chdev:</b>	Specifies the valid voice channel device handle obtained when the channel was opened using <b>dx_open( )</b> .
<b>ct_devinfo:</b>	Specifies a pointer to the data structure CT_DEVINFO that will contain the voice channel device information.

On return from the function, the CT\_DEVINFO structure contains the relevant information and is declared as follows:

```
typedef struct {
    unsigned long   ct_prodid;
    unsigned char   ct_devfamily;
    unsigned char   ct_devmode;
    unsigned char   ct_nettype;
    unsigned char   ct_busmode;
    unsigned char   ct_busencoding;
    unsigned char   ct_rfu[7];
} CT_DEVINFO;
```

***dx\_getctinfo( )*** returns information about a voice channel device handle

Valid values for each member of the CT\_DEVINFO structure are defined in *dxxxlib.h*.

<b>Field</b>	<b>Description</b>
<b>ct_prodid</b>	Contains a valid Dialogic product identification number for the device
<b>ct_devfamily</b>	Specifies the device family and may contain either: CT_DFD41E voice channel of a D/xxE board such as D/41ESC CT_DFSPAN voice channel of a DIALOG/HD high density board such as D/240SC, D/320SC, D/240SC-T1, D/300SC-E1, or D/160SC-LS board
<b>ct_devmode</b>	Specifies a device mode field that is valid only for the D/xxE board and may contain one of the following: CT_DMRESOURCE analog channel not in use CT_DMNETWORK analog channel available to process calls from the telephone network
<b>ct_nettype</b>	Specifies the type of network interface for the device. Valid values are: CT_NTNONE D/xxE board configured as a resource device; voice channels available for call processing; analog channels are disabled. CT_NTANALOG analog and voice devices on board are handling call processing CT_NTT1 T-1 digital channel CT_NTE1 E-1 digital channel
<b>ct_busmode</b>	Specifies the bus architecture used to communicate with other devices in the system: CT_BMSCBUS SCbus architecture
<b>ct_busencoding</b>	Describes the PCM encoding used on the bus. Valid values are: CT_BEULAW Mu-law encoding CT_BEALAW A-law encoding

**returns information about a voice channel device handle**      **dx\_getctinfo( )**

---

### ■ Cautions

This function will fail if an invalid voice channel device handle is specified.

### ■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <errno.h>
main()
{
    int chdev;                /* Channel device handle */
    CT_DEVINFO ct_devinfo;    /* Device information structure */
    /* Open board 1 channel 1 devices */
    if ((chdev = dx_open("dxxxB1C1", 0)) == -1) {
        printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
        exit(1);
    }
    /* Get Device Information */
    if (dx_getctinfo(chdev, &ct_devinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
    printf("%s Product Id = 0x%x, Family = %d, Mode = %d, Network = %d, Bus ...mode = %d,
    Encoding = %d", ATDV_NAMEP(chdev),
        ct_devinfo.ct_prodid, ...ct_devinfo.ct_devfamily, ct_devinfo.ct_devmode,
    ct_devinfo.ct_nettype, ...ct_devinfo.ct_busmode,
        ct_devinfo.ct_busencoding);
}
```

***dx\_getctinfo()***      ***returns information about a voice channel device handle***

---

■ **Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADEXTTS	SCbus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADTYPE	Invalid local time slot channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux System Error

■ **See Also**

- **ag\_getctinfo()**
- **dt\_getctinfo()**

returns SCbus time slot number

**dx\_getxmitslot( )**

---

**Name:** int dx\_getxmitslot(chdev, sc\_tsinfo)  
**Inputs:** int chdev • channel device handle  
SC\_TSINFO \*sc\_tsinfo • pointer to SCbus time slot  
information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** dxxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The [dx\\_getxmitslot\( \)](#) function returns SCbus time slot number of voice transmit channel. The SCbus time slot information is contained in an SC\_TSINFO structure that includes the number of the SCbus time slot connected to the voice transmit channel.

**NOTE:** SCbus convenience function [nr\\_scroute\( \)](#) includes [dx\\_getxmitslot\( \)](#) functionality. See the *SCbus Routing Guide* for more information on convenience functions.

Parameter	Description
<b>chdev:</b>	Specifies the voice channel device handle obtained when the channel was opened using <a href="#">dx_open( )</a> .
<b>sc_tsinfo:</b>	Specifies a pointer to the data structure SC_TSINFO. The SC_TSINFO structure is declared as follows: <pre>typedef struct {     unsigned long  sc_numts;     long          *sc_tsarray; } SC_TSINFO;</pre>
	The <b>sc_numts</b> member of the SC_TSINFO structure must be initialized with the number of SCbus time slots requested (1 for a voice channel). The <b>sc_tsarrayp</b> field of the SC_TSINFO structure must be initialized with a pointer to a valid array. Upon return from the function, the array will contain the number (between 0 and 1023) of the SCbus time slot on which the voice channel transmits. A voice channel on an SCbus based board can transmit on only one SCbus time slot.

### ■ Cautions

This function will fail when an invalid channel device handle is specified.

**`dx_getxmitslot()`**

*returns SCbus time slot number*

■ **Example**

```
#include <srllib.h>
#include <dxxxlib.h>
#include <errno.h>
main()
{
    int chdev;                /* Channel device handle */
    SC_TSINFO    sc_tsinfo;   /* Time slot information structure */
    long        scts;        /* SCbus time slot */
    /* Open board 1 channel 1 devices */
    if ((chdev = dx_open("dxxxBlC1", 0)) == -1) {
        printf("Cannot open channel dxxxBlC1.  errno = %d", errno);
        exit(1);
    }
    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarray = &scts;
    /* Get SCbus time slot connected to transmit of voice channel 1 on board ...1 */
    if (dx_getxmitslot(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
    printf("%s is transmitting on SCbus time slot %d", ATDV_NAMEP(chdev), ...scts);
}
```

■ **Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel is already disconnected from SCbus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux System Error

■ **See Also**

- **ag\_listen()**
- **dt\_listen()**
- **fx\_listen()**

---

***dx\_listen()*** ***connects voice listen channel to SCbus time slot***

---

**Name:** int dx\_listen(chdev, sc\_tsinfop)  
**Inputs:** int chdev • voice channel device handle  
SC\_TSINFO \*sc\_tsinfop • pointer to SCbus time slot information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** dxxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The **dx\_listen()** function [connects voice listen channel to SCbus time slot](#). This function uses the information stored in the SC\_INFO structure to connect the receive voice (listen) channel to an SCbus time slot.. This function sets up a half-duplex connection. For a full-duplex connection, the receive (listen) channel of the other device must be connected to the voice transmit channel.

**NOTE:** SCbus convenience function **nr\_scroute()** includes **dx\_listen()** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

<b>Parameter</b>	<b>Description</b>
<b>chdev:</b>	Specifies the voice channel device handle obtained when the channel was opened using <b>dx_open()</b> .
<b>sc_tsinfop:</b>	Specifies a pointer to the data structure SC_TSINFO.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {  
    unsigned long    sc_numts;  
    long            *sc_tsarray;  
} SC_TSINFO;
```

The **sc\_numts** field of the SC\_TSINFO structure must be set to 1. The **sc\_tsarray** field of the SC\_TSINFO structure must be initialized with a pointer to a valid array. The first element of this array must contain a valid SCbus time slot number (between 0 and 1023) which was obtained by issuing an **xx\_getxmitslot( )** function (xx = ag, dt, dx or fx). Upon return from the **dx\_listen( )** function, the voice receive channel will be connected to the SCbus time slot.

Although multiple voice channels may listen (be connected) to the same SCbus time slot, the receive of a voice channel can connect to only one SCbus time slot.

#### ■ Cautions

This function will fail when:

- An invalid channel device handle is specified.
- An invalid SCbus time slot number is specified.

**■ Example**

```
#include <srllib.h>
#include <dxxlib.h>
#include <errno.h>
main()
{
    int chdev;                /* Channel device handle */
    SC_TSINFO sc_tsinfo;     /* Time slot information structure */
    long scts;               /* SCbus time slot */
    /* Open board 1 channel 1 device */
    if ((chdev = dx_open("dxxxBlC1", 0)) == -1) {
        printf("Cannot open channel dxxxBlC1.  errno = %d", errno);
        exit(1);
    }
    /* Fill in the SCbus time slot information */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarrayp = &scts;
    /* Get SCbus time slot connected to transmit of analog channel 1 on board 1 */
    if (ag_getxmitslot(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
    /* Connect the receive of voice channel 1 on board 1 to SCbus time slot of analog
channel 1 */
    if (dx_listen(chdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function

**ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	SCbus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLTSCNCT	Channel is already connected to SCbus
EDX_SH_LIBBSY	Switch Handler library busy-
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux System Error-

**■ See Also**

- **ag\_getxmitslot()**
- **dt\_getxmitslot()**
- **fx\_getxmitslot()**
- **dx\_unlisten()**

**`dx_unlisten()`**

*disconnects voice receive channel from SCbus*

---

**Name:** int `dx_unlisten(chdev)`  
**Inputs:** int `chdev`                      • channel device handle  
**Returns:** 0 on success  
          -1 on error  
**Includes:** `dxxlib.h`  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The **`dx_unlisten()`** function *disconnects voice receive channel from SCbus*. This function disconnects the voice receive (listen) channel from the SCbus time slot. Calling the **`dx_listen()`** function to connect to a different SCbus time slot will automatically break an existing connection. Thus, when changing connections, you need not call the **`dx_unlisten()`** function.

**NOTE:** SCbus convenience function **`nr_scunroute()`** includes **`dx_unlisten()`** functionality. See the *SCbus Routing Guide* for more information on convenience functions.

<b>Parameter</b>	<b>Description</b>
<b><code>chdev:</code></b>	Specifies the voice channel device handle obtained when the channel was opened using <b><code>dx_open()</code></b> .

### ■ Cautions

This function will fail when an invalid channel device handle is specified.

■ **Example**

```
#include <srllib.h>
#include <dxlib.h>
#include <errno.h>
main()
{
    int chdev;                /* Voice Channel device handle */
    /* Open board 1 channel 1 device */
    if ((chdev = dx_open("dxxxB1C1", 0)) == -1) {
        printf("Cannot open channel dxxxB1C1.  errno = %d", errno);
        exit(1);
    }
    /* Disconnect receive of board 1, channel 1 from all SCbus time slots */
    if (dx_unlisten(chdev) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(chdev));
        exit(1);
    }
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

**Equate**`EDX_BADPARM``EDX_SH_BADCMD``EDX_SH_BADEXTTS``EDX_SH_BADINDX``EDX_SH_BADLCLTS``EDX_SH_BADMODE``EDX_SH_BADTYPE``EDX_SH_CMDBLOCK``EDX_SH_LCLDSCNCT``EDX_SH_LIBBSY``EDX_SH_LIBNOTINIT``EDX_SH_MISSING``EDX_SH_NOCLK``EDX_SYSTEM`**Returned When**

Parameter error

Command is not supported in current bus configuration

SCbus time slot is not supported at current clock rate

Invalid Switch Handler index number

Invalid channel number

Function not supported in current bus configuration

Invalid channel type (voice, analog, etc.)

Blocking command is in progress

Channel already disconnected from SCbus

Switch Handler library busy

Switch Handler library uninitialized

Switch Handler is not present

Switch Handler clock failback failed

Linux System Error

**■ See Also**

- `dx_listen()`

returns SCbus time slot number

**fx\_getxmitslot()**

---

**Name:** int fx\_getxmitslot(dev,sc\_tsinfo)  
**Inputs:** int dev • fax channel device handle  
SC\_TSINFO \*sc\_tsinfo • pointer to SCbus time slot information structure

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
dxxlib.h  
faxlib.h

**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The **fx\_getxmitslot()** function returns SCbus time slot number of the fax transmit channel. The SCbus time slot information is contained in an SC\_TSINFO structure that includes the number of the SCbus time slot connected to the fax transmit channel.

**NOTE:** SCbus convenience function **nr\_scroute()** includes **fx\_getxmitslot()** functionality. You can use **nr\_scroute()** to set up full duplex SCbus routing necessary for fax transfers. See the *SCbus Routing Software : Reference for Unix* for more information on convenience functions.

Parameter	Description
<b>dev:</b>	Specifies the channel device handle obtained when the fax device was opened using <b>fx_open()</b> .
<b>sc_tsinfo:</b>	Specifies a pointer to the data structure SC_TSINFO.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {
    unsigned long sc_numts;
    long *sc_tsarray;
} SC_TSINFO;
```

The **sc\_numts** member of the SC\_TSINFO structure must be initialized with the number of SCbus time slots requested (1 for a fax channel). The **sc\_tsarray** member of the SC\_TSINFO structure must be initialized with a pointer to a valid array. Upon return from the function, the array will contain the number (between 0 and 1023) of the SCbus time slot on which the fax channel transmits.

A fax channel can transmit on only one SCbus time slot.

**fx\_getxmitslot()**

*returns SCbus time slot number*

### ■ Cautions

This function will fail when an invalid fax channel device handle is specified.

### ■ Example

```
#include <errno.h>
#include <srllib.h>
#include <fcntl.h>
#include <faxlib.h>

main()
{
    int dev;                /* Fax channel device handle. */
    SC_TSINFO sc_tsinfo;   /* Timeslot information structure. */
    long scts;             /* SCbus time slots. */
    .
    .
    /* Open the fax channel resource device. */
    if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
        /* Error opening device. */
        printf("Error opening channel, errno = %d\n", errno);
        exit(1);
    }
    /* Fill in the SC_TSINFO structure time slot information. */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarray = &scts;
    /* Get fax device channel SCbus transmit time slot. */
    if (fx_getxmitslot(dev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(dev));
        exit(1);
    }

    printf("Fax channel is transmitting on SCbus time slot %d\n", scts);
    .
    .
}
```

### ■ Errors

If the function returns -1, use the SRL Standard Attribute function

**ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel is already disconnected from SCbus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux System Error

### ■ See Also

- **ag\_listen()**
- **dt\_listen()**
- **dx\_listen()**
- **fx\_listen()**

---

***fx\_listen()*** ***connects fax listen channel to SCbus time slot***

---

**Name:** int `fx_listen(dev,sc_tsinfo)`  
**Inputs:** int `dev` • fax channel device handle  
SC\_TSINFO \*`sc_tsinfo` • pointer to SCbus time slot information structure  
**Returns:** 0 on success  
-1 on error  
**Includes:** `srllib.h`  
`dxxlib.h`  
`faxlib.h`  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

■ **Description**

The `fx_listen()` function [connects fax listen channel to SCbus time slot](#). This function uses the information stored in the SC\_TSINFO structure to connect the fax receive (listen) channel to an SCbus time slot. This function sets up a half-duplex connection. For a full-duplex connection, the receive (listen) channel of the other device must be connected to the fax transmit channel.

Calling the `fx_listen()` function to connect to a different SCbus time slot will automatically break an existing connection. Thus, when changing connections, you need not call the `fx_unlisten()` function.

**NOTE:** SCbus convenience function `nr_scroute()` includes `fx_listen()` functionality. You can use `nr_scroute()` to set up full duplex SCbus routing necessary for fax transfers. See the *SCbus Routing Guide* for more information on convenience functions.

<b>Parameter</b>	<b>Description</b>
<b>dev:</b>	Specifies the valid fax channel device handle obtained when the channel was opened using <code>fx_open()</code> .
<b>sc_tsinfo:</b>	Specifies a pointer to the data structure SC_TSINFO.

The SC\_TSINFO structure is declared as follows:

```
typedef struct {
    unsigned long    sc_numts;
    long            *sc_tsarray;
} SC_TSINFO;
```

The **sc\_numts** member of the SC\_TSINFO structure must be set to 1. The **sc\_tsarray** field of the SC\_TSINFO structure must be initialized with a pointer to a valid array. The first element of this array must contain a valid SCbus time slot number (between 0 and 1023) which was obtained by issuing a **xx\_getxmitslot()** function (xx = ag, dt, dx or fx). Upon return from the **fx\_listen()** function, the fax receive channel is connected to this time slot. Although multiple SCbus device channels may listen (be connected) to the same SCbus time slot, the fax receive (listen) channel can connect to only one SCbus time slot.

#### ■ Cautions

This function will fail when:

- An invalid fax channel device handle is specified.
- An invalid SCbus time slot is specified.

### ■ Example

```

#include <errno.h>
#include <srllib.h>
#include <dxlib.h>
#include <faxlib.h>

#include "sctools.h"

main()
{
    int voxdev;          /* Voice channel device handle. */
    int dev;            /* Fax channel device handle. */
    SC_TSINFO sc_tsinfo; /* SCbus time slot information structure. */
    long scts;         /* SCbus time slot. */
    .
    .
    /* Open the fax channel device. */
    if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
        /* Error opening device. */
        printf("Error opening channel, errno = %d\n", errno);
        exit(1);
    }
    /* Open the VOICE channel device on the D/160SC-LS. */
    if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
        /* Error opening device. */
        printf("Error opening channel, errno = %d\n", errno);
        exit(1);
    }
    .
    .
    /*
    * Break the full-duplex connection between the Voice
    * channel device and the Network analog device.
    * Use the SCbus routing convenience function nr_scuroute().
    */
    if (nr_scuroute(voxdev, SC_VOX, voxdev, SC_LSI, SC_FULLDUP) == -1) {
        /* Error during SCbus unrouting. */
        printf("Error unrouting channel\n");
        printf("Error - %s (error code %d)\n",
            ATDV_ERRMSGP(voxdev), ATDV_LASTERR(voxdev));
        if (ATDV_LASTERR(voxdev) == EDX_SYSTEM) {
            printf("errno = %d\n", errno);
        }
    }
    /*
    * Set full-duplex connection between the fax
    * channel device and the Network analog device.
    */

    /* Fill in the SC_TSINFO structure time slot information. */
    sc_tsinfo.sc_numts = 1;
    sc_tsinfo.sc_tsarray = &scts;

    /* Get Network analog device's SCbus transmit time slot. */
    if (ag_getxmitslot(voxdev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(voxdev));
        exit(1);
    }
    /*
    * Connect the fax channel to "listen" to the Network
    * channel's SCbus transmit time slot. Pass the time slot

```

```

    * information in the SC_TSINFO structure to fx_listen().
    */
    if (fx_listen(dev, &sc_tsinfo) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(dev));
        exit(1);
    }
    .
    .
    /* Complete full-duplex connection between the fax channel device
    * and the Network channel device using fx_getxmitslot()
    * and ag_listen().
    */
    .
    .
    /* Call fax API functions for fax transfers. */
    .
    .

```

### ■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. The error codes returned by **ATDV\_LASTERR()** are:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	SCbus time slot is not supported at current clock rate
EDX_SH_BADINDEX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLTSCNCT	Channel is already connected to SCbus
EDX_SH_LIBBSY	Switch Handler library busy-
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Linux System Error-

*fx\_listen()*

*connects fax listen channel to SCbus time slot*

---

■ See Also

- `ag_getxmitslot()`
- `dt_getxmitslot()`
- `dx_getxmitslot()`
- `fx_getxmitslot()`
- `fx_unlisten()`

---

**Name:** int fx\_unlisten(dev)  
**Inputs:** int dev • fax channel device handle  
**Returns:** 0 on success  
           -1 on failure  
**Includes:** srllib.h  
               dxxlib.h  
               faxlib.h  
**Category:** SCbus Routing  
**Mode:** Synchronous

---

### ■ Description

The **fx\_unlisten( )** function disconnects fax listen channel from SCbus. This function disconnects a fax receive (listen) channel from the SCbus transmit time slot.

Calling the **fx\_listen( )** function to connect to a different SCbus time slot will automatically break an existing connection. Thus, when changing connections, you need not call the **fx\_unlisten( )** function.

**NOTE:** The SCbus convenience function **nr\_scunroute( )** includes **fx\_unlisten( )** functionality. You can use **nr\_scunroute( )** to break the full duplex SCbus routing necessary for fax transfers. See the *SCbus Routing Software Reference for Unix* for more information on convenience functions.

Parameter	Description
<b>dev:</b>	Specifies the fax channel device handle obtained when the channel was opened using <b>fx_open( )</b> .

### ■ Cautions

This function will fail when an invalid fax channel device handle is specified.

**■ Example**

```
#include <errno.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main()
{
    int dev;          /* Fax channel device handle. */
    .
    .
    /* Open the fax channel resource. */
    if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
        /* Error opening device. */
        printf("Error opening channel, errno = %d\n", errno);
        exit(1);
    }
    .
    .
    /*
     * Disconnect the fax channel device from "listening" to an
     * SCbus transmit time slot.
     */
    if (fx_unlisten(dev) == -1) {
        printf("Error message = %s", ATDV_ERRMSGP(dev));
        exit(1);
    }
    .
    .
}
```

**■ Errors**

If the function returns -1, use the SRL Standard Attribute function

**ATDV\_LASTERR()** to obtain the error code or use **ATDV\_ERRMSGP()** to obtain a descriptive error message. One of the following error codes may be returned:

<b>Equate</b>	<b>Returned When</b>
EDX_BADPARAM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	SCbus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel already disconnected from SCbus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock failback failed
EDX_SYSTEM	Linux System Error

**■ See Also**

- **fx\_listen()**

*fx\_unlisten()*

*disconnects fax listen channel from SCbus*

---

## 4. Data Structure Reference

---

The data structures used by individual SCbus routing functions are described in this chapter. These structures are used to control the operation of functions and to return information. The data structures are defined in header files located in */usr/include/dialogic*.

The SCbus data structures include:

- Channel/Time Slot Device Information (CT\_DEVINFO) structure that contains device (analog, digital, fax, voice, etc.) and device configuration information. The **xx\_getctinfo( )** functions get the device information to fill this structure (xx = ag, dt, dx or fx). This data structure is defined in *dxxxlib.h* and *dtilib.h*.
- SCbus Time Slot Information (SC\_TSINFO) structure that contains the numeric quantity of SCbus time slots (typically 1) assigned to a device and a pointer to an array that contains the SCbus time slot number(s). The **xx\_getxmitslot( )** functions get the information to fill the data structure (xx = ag, dt, dx or fx). Then the **xx\_listen( )** functions use this information to connect two SCbus devices. This data structure is defined in *dxxxlib.h* and *scroute.h*.

### 4.1. Channel/Time Slot Device Information (CT\_DEVINFO)

The CT\_DEVINFO structure supplies information about a device. This structure is used by the SCbus routing functions identified by the suffix **getctinfo( )**. On return from the function, the CT\_DEVINFO structure contains the relevant information and is defined as follows:

```
typedef struct {
    unsigned long    ct_prodid;
    unsigned char   ct_devfamily;
    unsigned char   ct_devmode;
    unsigned char   ct_nettype;
    unsigned char   ct_busmode;
    unsigned char   ct_busencoding;
    unsigned char   ct_rfu[7];           /* reserved for future use */
} CT_DEVINFO;
```

Valid values for each member of the CT\_DEVINFO structure are:

## SCbus Routing Software Reference for Linux

Field	Description
<b>ct_prodid</b>	Contains a valid Dialogic product identification number for the device [length: 4 (unsigned long)]
<b>ct_devfamily</b>	Specifies the device family [length: 1 (unsigned char)] and may contain either: CT_DFD41E            analog or voice channel of a D/xxE board such as D/41ESC or VFX/40ESC CT_DFSPAN            analog channel such as on a D/160SC-LS board, a voice channel such as on a D/240SC, D/320SC, D/240SC-T1, D/300SC-E1 or D/160SC-LS board, or a digital channel such as on a D/240SC-T1 or D/300SC-E1 board
<b>ct_devmode</b>	Specifies a device mode field [length: 1 (unsigned char)] that is valid only for the D/xxE board such as D/41ESC or VFX/40ESC and may contain either: CT_DMRESOURCE      analog channel not in use CT_DMNETWORK        analog channel available to process calls from the telephone network
<b>ct_nettype</b>	Specifies the type of network interface for the device [length: 1 (unsigned char)]. Valid values are: CT_NTNONE            D/xxE board such as D/41ESC or VFX/40ESC configured as a resource device; voice channels available for call processing; analog channels are disabled. CT_NTANALOG          analog and voice devices on board are handling call processing CT_NTT1                T-1 digital network interface CT_NTE1                E-1 digital network interface
<b>ct_busmode</b>	Specifies the bus architecture used to communicate with other devices in the system [length: 1 (unsigned char)] CT_BMSCBUS            SCbus architecture

Field	Description
<b>ct_busencoding</b>	Describes the PCM encoding used on the bus [length: 1 (unsigned char)]. Valid values are: CT_BEULAW            Mu-law encoding CT_BEALAW            A-law encoding

## 4.2. SCbus Time Slot Information (SC\_TSINFO)

The SC\_TSINFO structure contains the numeric quantity defining how many SCbus time slots are associated with a particular device (typically 1) and a pointer to an array that will hold the actual SCbus time slot number(s) (valid numbers are 0 to 1023). The SC\_TSINFO structure is used by the SCbus routing functions identified by the suffix:

- **getxmitslot()** to supply SCbus time slot information about a device and
- **listen()** to use this time slot information to connect two SCbus devices.

The SC\_TSINFO structure is defined as follows:

```
typedef struct{
    unsigned long    sc_numts;
    long far        *sc_tsarray;
}SC_TSINFO;
```

The **sc\_numts** field of the SC\_TSINFO structure must be initialized with the number of SCbus time slots, typically 1. The **sc\_tsarray** field must be initialized with a pointer to an array of long integers.

*SCbus Routing Software Reference for Linux*

## 5. Demo Program

---

The Linux demo program illustrates how SCbus routing functions connect various SCbus devices. The source code and demo explanation are presented in this chapter.

### 5.1. Answering Machine Demo

The answering machine demo program, *pansr.c*, is presented in this section and can be found in the *usr/dialogic/dx\_demos/ansr* directory of the voice software. This demo illustrates the use of the SCbus convenience functions, **nr\_scroute( )** and **nr\_scunroute( )**, to connect voice resources and analog loop-start network interfaces or T-1/E-1 digital network interfaces via the SCbus.

This demo program provides answering machine functionality for up to 12 analog or digital channels on SCbus-based board(s). The term **frontend** used in the demo refers to the analog device or T-1/E-1 device that interfaces to the telephone network.

The demo determines whether the telephone network interface is an analog device, a digital T-1 device or a digital E-1 device based on your input (see demo code for options). Having determined the type of telephone network interface, the demo opens the appropriate analog or digital channels and voice channels.

Initially, the demo disconnects the network and resource devices that will be used by the demo from the SCbus by calling the **nr\_scunroute( )** function. Then the demo calls the **nr\_scroute( )** function to make a full-duplex connection between the network interface channels and the voice channels via the SCbus.

The demo initializes the channel states to detect an incoming call and then polls for a call. When a call is detected, the demo answers and plays a introductory message (*intro.vox*), beeps, records up to 10 seconds of an incoming message in a voice (VOX) file and hangs up. To playback recorded messages, you call into each channel, enter an access code and the demo plays back all messages stored for that channel.

```
/******  
* Multi-line Asynchronous Answering Phone Demo Program *  
*****  
*  
* P P P P P A A N N S S S S R R R R R C C C C *  
* P P A A N N S R R C C *  
* P P A A N N S S S S R R C *  
* P P P P P A A A A A N N N S R R R R R . . . C *  
* P A A N N S S R R . . . C C *  
* P A A N N S S S S R R . . . C C C C *  
*****
```

## SCbus Routing Software Reference for Linux

```
* Copyright (c) 1997 by Dialogic Corp. All Rights Reserved *
*****/

/**
** System Header Files
**/
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

/**
** Dialogic Header Files
**/
#include <srllib.h>
#include <dxlib.h>
#include <dtilib.h>
#include <errno.h>

#include "sctools.h"

#include "answer.h"

/*
 * Globals
 */
char tmpbuff[ 256 ]; /* Temporary Buffer */

/**
** Data structure which defines the states for each channel
**/
typedef struct dx_info {
    int chdev; /* Channel Device Descriptor */
    int tsdev; /* Timeslot Device Descriptor */
    int state; /* State of Channel */
    int msg_fd; /* File Desc for Message Files */
    DV_DIGIT digbuf; /* Buffer for DTMF Digits */
    DX_IOTT iott[ 1 ]; /* I/O Transfer Table */
    char msg_name[ MAXMSG+1 ]; /* Message Filename */
    char ac_code[ MAXDTMF+1 ]; /* Access Code for this Channel */
} DX_INFO;

DX_INFO dxinfo[ MAXCHANS+1 ];

/**
** Global data
**/

/*
 * File Descriptors for VOX Files
 */
int introfd;
int invalidfd;
int goodbyefd;

int maxchans = 4; /* Default Number of D/4x Channels to use */
int d4xbdnum = 1; /* Default D/4x Board Number to Start use */
int dtibdnum = 1; /* Default DTI Board Number to Start use */
int frontend = CT_NIT1; /* Default network frontend is T1 */
```

## 5. Demo Program

```
int scbus = TRUE; /* Default Bus mode is SBus */
int routeag = FALSE; /* Route analog frontend to resource ??? */

/*
 * Externals
 */
extern int errno;
extern char * optarg; /* Pointer to Argument Parameter (getopt) */
extern int optind; /* Next Index into argv[] (getopt) */

/*****
 * NAME: void intr_hdlr()
 * DESCRIPTION: Handler called when one of the following signals is
 * received: SIGHUP, SIGINT, SIGQUIT, SIGTERM.
 * This function stops I/O activity on all channels and
 * closes all the channels.
 * INPUT: None
 * OUTPUT: None.
 * RETURNS: None.
 * CAUTIONS: None.
 *****/
void intr_hdlr()
{
    int channum;

    disp_msg( "Process Terminating ..." );

    /*
     * Close all the channels opened after stopping all I/O activity.
     * It is okay to stop the I/O on a channel as the program is
     * being terminated.
     */
    for ( channum = 1; channum <= maxchans; channum++ ) {
        set_hkstate( channum, DX_ONHOOK );
        if (frontend == CT_NTANALOG) {
            /*
             * If analog frontend timeslots had been routed to the resource,
             * then unrout them.
             */
            if ((scbus == TRUE) && (routeag == TRUE)) {
                nr_scunroute( dxinfo[ channum ].chdev, SC_LSI,
                             dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP );
            }
        } else { /* Digital Frontend */
            /*
             * Unroute the digital timeslots from their resource channels.
             */
            if (scbus == TRUE) {
                nr_scunroute( dxinfo[ channum ].tsdev, SC_DTI,
                             dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP );
            }
        }
        /*
         * If E1 frontend, reset to "Blocking state"
         */
        if (frontend == CT_NTE1) {
            dt_settssigsim( dxinfo[ channum ].tsdev, DTB_BON );
        }
        dt_close( dxinfo[ channum ].tsdev );
    }
}
```

## SCbus Routing Software Reference for Linux

```
        dx_close( dxinfo[ channum ].chdev );
    }
    QUIT( 0 );
}

/*****
 *      NAME: void chkargs( argc, argv )
 * DESCRIPTION: Check Command Line Arguments, Display Help or Set
 *              globals to indicate board number and number of
 *              channels to use.
 *      INPUT: int argc; - Argument Count.
 *              char *argv[]; - Array of Pointers to Arguments.
 *      OUTPUT: None.
 *      RETURNS: None.
 *      CAUTIONS: None.
 *****/
void chkargs( argc, argv )
    int argc;
    char *argv[];
{
    int arg;

    /*
     * Check Command Line for Options
     * N.B. Arguments to Options MUST be Separated by White Space
     */
    while ( ( arg = getopt( argc, argv, "d:t:n:f:pr?" ) ) != -1 ) {
        switch ( arg ) {
            case 'd':
                /*
                 * First D/4x Board Number
                 */
                d4xbdnum = (int) atol( optarg );
                break;

            case 't':
                /*
                 * First DTI Board Number
                 */
                dtibdnum = (int) atol( optarg );
                break;

            case 'n':
                /*
                 * Number of Channels to Use
                 */
                maxchans = (int) atol( optarg );
                break;

            case 'f':
                /*
                 * Frontend - T1, E1.
                 */
                if (*optarg == 'T') {
                    frontend = CT_NTTL;
                } else if (*optarg == 'E') {
                    frontend = CT_NTEL;
                } else if (*optarg == 'A') {
                    frontend = CT_NTANALOG;
                }
            }
        }
    }
}
```

## 5. Demo Program

```
    } else {
        printf( "Unknown frontend value: %s\n", optarg );
        exit( 1 );
    }
    break;

case '?':
    /*
     * Display Help Message
     */
    printf( "\nUsage: %s [-d d4xnum] [-t dtinum] [-n N] [-f frontend]\n", argv[ 0 ]
);
    printf("-d d4xnum\tNumber of the first D/4x Board to use\n" );
    printf("-t dtinum\tNumber of the first DTI Board to use\n" );
    printf("-n N\t\tNumber of D/4x channels to use, max: %d\n", MAXCHANS );
    printf("-f frontend\tAnalog, T1 or E1\n" );
    exit ( 1 );
}
}

/*****
 * NAME: void sysinit()
 * DESCRIPTION: Start D/4x System, Enable CST events, put line ON-Hook
 * and open VOX files.
 * INPUT: None.
 * OUTPUT: None.
 * RETURNS: None.
 * CAUTIONS: None.
 *****/
void sysinit()
{
    int  channum;
    char d4xname[ 32 ];
    char dtiname[ 32 ];
    char *chname;
    int  chno;
    int  bddev;

    if ( maxchans > MAXCHANS ) {
        sprintf( tmpbuff, "Only %d Channels will be used", MAXCHANS );
        disp_msg( tmpbuff );
        maxchans = MAXCHANS;
    }

    disp_msg( "Initializing ...." );

    /*
     * Open VOX Files
     */
    if ( ( introfd = open( INTRO_VOX, O_RDONLY ) ) == -1 ) {
        sprintf( tmpbuff, "Cannot open %s", INTRO_VOX );
        disp_msg( tmpbuff );
        QUIT( 1 );
    }

    if ( ( invalidfd = open( INVALID_VOX, O_RDONLY ) ) == -1 ) {
        sprintf( tmpbuff, "Cannot open %s", INVALID_VOX );
        disp_msg( tmpbuff );
        QUIT( 1 );
    }
}
```

## SCbus Routing Software Reference for Linux

```
}

if ( ( goodbyefd = open( GOODBYE_VOX, O_RDONLY ) ) == -1 ) {
    sprintf( tmpbuff, "Cannot open %s", GOODBYE_VOX );
    disp_msg( tmpbuff );
    QUIT( 1 );
}

/*
 * Clear the dxinfo structure
 * Initialize Channel States to Detect Call.
 */
memset( dxinfo, 0, (sizeof( DX_INFO ) * (MAXCHANS+1)) );

for ( channum = 1; channum <= maxchans; channum++ ) {
    /*
     * Open the D/4x Channels
     */
    sprintf( d4xname, "dxxxB%dC%d",
            (channum % 4) ? (channum / 4) + d4xbdnum :
            d4xbdnum + (channum / 4) - 1,
            (channum % 4) ? (channum % 4) : 4 );

    if ( ( dxinfo[ channum ].chdev = dx_open( d4xname, 0 ) ) == -1 ) {
        sprintf( tmpbuff, "Unable to open channel %s, errno = %d",
                d4xname, errno );
        disp_msg( tmpbuff );
        QUIT( 2 );
    }

    if (frontend == CT_NTANALOG) {
        /*
         * Route analog frontend timeslots to its resource in SCbus mode,
         * if required.
         */
        if ((scbus == TRUE) && (routeag == TRUE)) {
            nr_scunroute( dxinfo[ channum ].chdev, SC_LSI,
                          dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP );
            if (nr_scroute( dxinfo[ channum ].chdev, SC_LSI,
                            dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP )
                == -1) {
                sprintf( tmpbuff, "nr_scroute() failed for %s",
                        ATDV_NAMEP( dxinfo[ channum ].chdev ) );
                disp_msg( tmpbuff );
                QUIT( 2 );
            }
        }
    }
} else { /* Digital Frontend */
    /*
     * Form digital timeslots' names based upon bus mode.
     */
    if (scbus == TRUE) {
        sprintf( dtiname, "dtiB%dT%d", dtibdnum, channum );
    } else {
        sprintf( dtiname, "/dev/dtiB%dT%d", dtibdnum, channum );
    }

    /*
     * Open DTI timeslots.
     */
    if ( ( dxinfo[ channum ].tsdev = dt_open( dtiname, 0 ) ) == -1 ) {
        sprintf( tmpbuff, "Unable to open timeslot %s, errno = %d",
                dtiname, errno );
    }
}
}
```

## 5. Demo Program

```
        dtiname, errno );
disp_msg( tmpbuff );
QUIT( 2 );
}

/*
 * Route timeslots to channels based upon bus mode.
 */
if ( scbus != TRUE ) {
    sprintf( tmpbuff, "Bus type must be SCbus only!\n" );
    disp_msg( tmpbuff );
    QUIT( 2 );
}

nr_scuroute( dxinfo[ channum ].tsdev, SC_DTI,
             dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP );
if ( nr_scroute( dxinfo[ channum ].tsdev, SC_DTI,
                dxinfo[ channum ].chdev, SC_VOX, SC_FULLDUP )
    == -1 ) {
    sprintf( tmpbuff, "nr_scroute() failed for %s - %s",
            ATDV_NAMEP( dxinfo[ channum ].chdev ),
            ATDV_NAMEP( dxinfo[ channum ].tsdev ) );
    /* disp_msg( tmpbuff );
       sprintf( tmpbuff, "dxdev: %s dtdev: %s\n",
               ATDV_ERRMSGP( dxinfo[ channum ].chdev ),
               ATDV_ERRMSGP( dxinfo[ channum ].tsdev ) );
       disp_msg( tmpbuff );
       */
    QUIT( 2 );
}

/*
 * Enable the CST events
 */
if ( dx_setevtmsk( dxinfo[ channum ].chdev, DM_RINGS ) == -1 ) {
    sprintf( tmpbuff, "Cannot set CST events for %s",
            ATDV_NAMEP( dxinfo[ channum ].chdev ) );
    disp_msg( tmpbuff );
    disp_err( channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state );
    QUIT( 2 );
}

/*
 * Set to answer after MAXRING rings
 */
if ( dx_setstrings( dxinfo[ channum ].chdev, MAXRING ) == -1 ) {
    sprintf( tmpbuff, "dx_setstrings() failed for %s",
            ATDV_NAMEP( dxinfo[ channum ].chdev ) );
    disp_msg( tmpbuff );
    disp_err( channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state );
    QUIT( 2 );
}

sprintf( dxinfo[ channum ].ac_code, "%d234", channum );
sprintf( dxinfo[ channum ].msg_name, "message%d.vox", channum );

/*
 * If it is a digital network environment, disable idle on the timeslot,
 * set it to signalling insertion and set the signalling event mask.
 */
if ( frontend != CT_NTANALOG ) {
```

## SCbus Routing Software Reference for Linux

```
if ( dt_setidle( dxinfo[ channum ].tsdev, DTIS_DISABLE ) == -1 ) {
    sprintf( tmpbuff, "Cannot disable IDLE for %s",
            ATDV_NAMEP( dxinfo[ channum ].tsdev ) );
    disp_msg( tmpbuff );
    disp_err( channum, dxinfo[ channum ].tsdev, dxinfo[ channum ].state );
    QUIT( 2 );
}

if ( dt_setsigmod( dxinfo[ channum ].tsdev, DIM_SIGINS ) == -1 ) {
    sprintf( tmpbuff, "Cannot set SIGINS for %s",
            ATDV_NAMEP( dxinfo[ channum ].tsdev ) );
    disp_msg( tmpbuff );
    disp_err( channum, dxinfo[ channum ].tsdev, dxinfo[ channum ].state );
    QUIT( 2 );
}

/*
 * Unblock E1 timeslot's signalling bits to ready state.
 */
if ( frontend == CT_NTE1 ) {
    if ( dt_settssigsim( dxinfo[ channum ].tsdev,
                        DTB_DON | DTB_COFF | DTB_BOFF | DTB_AON ) == -1 ) {
        sprintf( tmpbuff, "Cannot set bits to ready state on %s",
                ATDV_NAMEP( dxinfo[ channum ].tsdev ) );
        disp_msg( tmpbuff );
        disp_err( channum, dxinfo[ channum ].tsdev,
                dxinfo[ channum ].state );
        QUIT( 2 );
    }
}

if ( dt_setevtmsk( dxinfo[ channum ].tsdev, DTG_SIGEVT,
                  DIMM_AOFF | DIMM_AON, DTA_SEIMSK ) == -1 ) {
    disp_msg( "Unable to set DTI signalling event mask" );
    QUIT( 2 );
}

/*
 * Start the application by putting the channel to ON-HOOK state.
 */
dxinfo[ channum ].state = ST_ONHOOK;
set_hkstate( channum, DX_ONHOOK );

if ( frontend != CT_NTANALOG ) {
    disp_status( channum, "Ready to accept a call" );
}

/*
 * Display number of channels being used
 */
sprintf( tmpbuff, "Using %d line%s", maxchans, maxchans > 1 ? "s" : "" );
disp_msg( tmpbuff );
}

/*****
 * NAME: int play( channum, filedesc )
 * DESCRIPTION: Set up IOTT and TPT's and Initiate the Play-Back
 * INPUT: int channum; - Index into dxinfo structure
 * int filedesc; - File Descriptor of VOX file to Play-Back
 *****/
```

## 5. Demo Program

```

*      OUTPUT: Starts the play-back
*      RETURNS: -1 = Error
*              0 = Success
*      CAUTIONS: None
*****
int play( channum, filedesc )
    int channum;
    int filedesc;
{
    DV_TPT  tpt[ 2 ];

    /*
     * Rewind the file
     */
    if ( lseek( filedesc, 0, SEEK_SET ) == -1 ) {
        sprintf( tmpbuff, "Cannot seek to the beginning of the VOX file",
            ATDV_NAMEP( dxinfo[ channum ].chdev ) );
        disp_msg( tmpbuff );
    }

    /*
     * Clear and Set-Up the IOTT structure
     */
    memset( dxinfo[ channum ].iott, 0, sizeof( DX_IOTT ) );

    dxinfo[ channum ].iott[ 0 ].io_type = IO_DEV | IO_EOT;
    dxinfo[ channum ].iott[ 0 ].io_fhandle = filedesc;
    dxinfo[ channum ].iott[ 0 ].io_length = -1;

    /*
     * Clear and then Set the DV_TPT structures
     */
    memset( tpt, 0, (sizeof( DV_TPT ) * 2) );

    /* Terminate Play on Receiving any DTMF tone */
    tpt[ 0 ].tp_type = IO_CONT;
    tpt[ 0 ].tp_termo = DX_MAXDTMF;
    tpt[ 0 ].tp_length = 1;
    tpt[ 0 ].tp_flags = TF_MAXDTMF;

    /* Terminate Play on Loop Current Drop */
    tpt[ 1 ].tp_type = IO_EOT;
    tpt[ 1 ].tp_termo = DX_LCOFF;
    tpt[ 1 ].tp_length = 1;
    tpt[ 1 ].tp_flags = TF_LCOFF;

    /*
     * Play VOX File on D/4x Channel, Normal Play Back
     */
    return( dx_play( dxinfo[ channum ].chdev, dxinfo[ channum ].iott,
        tpt, EV_ASYNC ) );
}

/*****
*      NAME: int record( channum, filedesc )
*      DESCRIPTION: Set up IOTT and TPT's and Initiate the record
*      INPUT: int channum; - Index into dxinfo structure
*            int filedesc; - File Descriptor of VOX file to Record to
*      OUTPUT: Starts the Recording
*      RETURNS: -1 = Error
*              0 = Success
*****/

```

## SCbus Routing Software Reference for Linux

```
* CAUTIONS: None
*****
int record( channum, filedesc )
    int channum;
    int filedesc;
{
    DV_TPT  tpt[ 4 ];

    /*
     * Clear and Set-Up the IOTT structure
     */
    memset( dxinfo[ channum ].iott, 0, sizeof( DX_IOTT ) );

    dxinfo[ channum ].iott[ 0 ].io_type = IO_DEV | IO_EOT;
    dxinfo[ channum ].iott[ 0 ].io_fhandle = filedesc;
    dxinfo[ channum ].iott[ 0 ].io_length = -1;

    /*
     * Clear and then Set the DV_TPT structures
     */
    memset( tpt, 0, (sizeof( DV_TPT ) * 4) );

    /* Terminate Record on Receiving any DTMF tone */
    tpt[ 0 ].tp_type = IO_CONT;
    tpt[ 0 ].tp_termno = DX_MAXDTMF;
    tpt[ 0 ].tp_length = 1;
    tpt[ 0 ].tp_flags = TF_MAXDTMF;

    /* Terminate Record on Loop Current Drop */
    tpt[ 1 ].tp_type = IO_CONT;
    tpt[ 1 ].tp_termno = DX_LCOFF;
    tpt[ 1 ].tp_length = 1;
    tpt[ 1 ].tp_flags = TF_LCOFF;

    /* Terminate Record on 5 Seconds of Silence */
    tpt[ 2 ].tp_type = IO_CONT;
    tpt[ 2 ].tp_termno = DX_MAXSIL;
    tpt[ 2 ].tp_length = 50;
    tpt[ 2 ].tp_flags = TF_MAXSIL;

    /* Terminate Record After 10 Seconds of Recording */
    tpt[ 3 ].tp_type = IO_EOT;
    tpt[ 3 ].tp_termno = DX_MAXTIME;
    tpt[ 3 ].tp_length = 100;
    tpt[ 3 ].tp_flags = TF_MAXTIME;

    /*
     * Record VOX File on D/4x Channel
     */
    return( dx_rec( dxinfo[ channum ].chdev, dxinfo[ channum ].iott,
        tpt, RM_TONE | EV_ASYNC ) );
}

/*****
 * NAME: int get_digits( channum, digbufp )
 * DESCRIPTION: Set up TPT's and Initiate get-digits function
 * INPUT: int channum; - Index into dxinfo structure
 * DV_DIGIT *digbufp; - Pointer to Digit Buffer
 * OUTPUT: Starts to get the DTMF Digits
 * RETURNS: -1 = Error
 *          0 = Success
 *****/
```

## 5. Demo Program

```
* CAUTIONS: None
*****
int get_digits( channum, digbufp )
    int channum;
    DV_DIGIT * digbufp;
{
    DV_TPT  tpt[ 3 ];

    /*
     * Clear and then Set the DV_TPT structures
     */
    memset( tpt, 0, (sizeof( DV_TPT ) * 3) );

    /* Terminate GetDigits on Receiving MAXDTMF Digits */
    tpt[ 0 ].tp_type = IO_CONT;
    tpt[ 0 ].tp_termo = DX_MAXDTMF;
    tpt[ 0 ].tp_length = MAXDTMF;
    tpt[ 0 ].tp_flags = TF_MAXDTMF;

    /* Terminate GetDigits on Loop Current Drop */
    tpt[ 1 ].tp_type = IO_CONT;
    tpt[ 1 ].tp_termo = DX_LCOFF;
    tpt[ 1 ].tp_length = 1;
    tpt[ 1 ].tp_flags = TF_LCOFF;

    /* Terminate GetDigits after 5 Seconds */
    tpt[ 2 ].tp_type = IO_EOT;
    tpt[ 2 ].tp_termo = DX_MAXTIME;
    tpt[ 2 ].tp_length = 50;
    tpt[ 2 ].tp_flags = TF_MAXTIME;

    return( dx_getdig( dxinfo[ channum ].chdev, tpt, digbufp, EV_ASYNC ) );
}

/*****
 * NAME: int set_hkstate( channum, state )
 * DESCRIPTION: Set the channel to the appropriate hook status
 * INPUT: int channum; - Index into dxinfo structure
 *        int state; - State to set channel to
 * OUTPUT: None.
 * RETURNS: -1 = Error
 *          0 = Success
 * CAUTIONS: None.
*****
int set_hkstate( channum, state )
    int channum;
    int state;
{
    int chdev = dxinfo[ channum ].chdev;
    int tsdev = dxinfo[ channum ].tsdev;

    /*
     * Make sure you are in CS_IDLE state before setting the
     * hook status
     */
    if ( ATDX_STATE( chdev ) != CS_IDLE ) {
        dx_stopch( chdev, EV_ASYNC );
        while ( ATDX_STATE( chdev ) != CS_IDLE );
    }

    switch ( frontend ) {
```

## SCbus Routing Software Reference for Linux

```
case CT_NTANALOG:
    if ( dx_sethook( chdev, (state == DX_ONHOOK) ? DX_ONHOOK : DX_OFFHOOK,
                    EV_ASYNC ) == -1 ) {
        disp_err( channum, chdev, dxinfo[ channum ].state );
        sprintf( tmpbuff, "Cannot set channel %s to %s-Hook (%s)",
                 ATDV_NAMEP( chdev ), (state == DX_ONHOOK) ? "On" : "Off",
                 ATDV_ERRMSGP( chdev ) );
        disp_msg( tmpbuff );
        return( -1 );
    }
    break;

case CT_NIT1:
    if ( dt_settssigsim( tsdev, (state == DX_ONHOOK) ? DTB_AOFF | DTB_BOFF :
                        DTB_AON | DTB_BON ) == -1 ) {
        disp_err( channum, tsdev, dxinfo[ channum ].state );
        sprintf( tmpbuff, "Cannot set bits to %s on %s (%s)",
                 (state == DX_ONHOOK) ? "AOFF-BOFF" : "AON-BON", ATDV_NAMEP( tsdev ),
                 ATDV_ERRMSGP( tsdev ) );
        disp_msg( tmpbuff );
        return( -1 );
    }
    break;

case CT_NIE1:
    if ( dt_settssigsim( tsdev, (state == DX_ONHOOK) ? DTB_AON : DTB_AOFF )
        == -1 ) {
        disp_err( channum, tsdev, dxinfo[ channum ].state );
        sprintf( tmpbuff, "Cannot set bits to %s on %s (%s)",
                 (state == DX_ONHOOK) ? "AON-BOFF" : "AOFF", ATDV_NAMEP( tsdev ),
                 ATDV_ERRMSGP( tsdev ) );
        disp_msg( tmpbuff );
        return( -1 );
    }
}

if ( state == DX_ONHOOK ) {
    if ( dx_clrdigbuf( chdev ) == -1 ) {
        sprintf( tmpbuff, "Cannot clear DTMF Buffer for %s",
                 ATDV_NAMEP( chdev ) );
        disp_msg( tmpbuff );
        disp_err( channum, chdev, dxinfo[ channum ].state );
    }
}

if ( frontend != CT_NTANALOG ) {
    switch ( state ) {
    case DX_ONHOOK:
        dxinfo[ channum ].state = ST_WTRING;
        break;

    case DX_OFFHOOK:
        dxinfo[ channum ].state = ST_INTRO;

        if ( play( channum, introfd ) == -1 ) {
            sprintf( tmpbuff, "Error playing Introduction on channel %s",
                     ATDV_NAMEP( chdev ) );
            disp_msg( tmpbuff );
        }
        break;
    }
}
}
```

## 5. Demo Program

```
    return( 0 );
}

/*****
 *      NAME: int curr_state( channum, event )
 * DESCRIPTION: Complete Processing of the Current State
 *              Identify the Next State to be Entered
 *      INPUT: int channum; - Index into dxinfo structure
 *              int event; - Event being processed
 *      OUTPUT: None
 *      RETURNS: Next Channel State
 *      CAUTIONS: None
 *****/
int curr_state( channum, event )
int channum;
int event;
{
    switch ( dxinfo[ channum ].state ) {
    case ST_WIRING:
        return( ST_OFFHOOK );

    case ST_OFFHOOK:
        return( ST_INTRO );

    case ST_INTRO:
        if ( event == TDX_PLAY &&
            (ATDX_TERMMSK( dxinfo[ channum ].chdev ) & TM_MAXDTMF ) ) {
            return( ST_GETDIGIT );
        }

        return( ST_RECORD );

    case ST_GETDIGIT:
        /*
         * Validate Digits for the Channel
         */
        if ( strcmp( dxinfo[ channum ].ac_code,
                    dxinfo[ channum ].digbuf.dg_value ) == 0 ) {
            /*
             * Correct Digits Entered
             */
            return( ST_PLAY );
        }

        /*
         * Invalid Digits Entered
         */
        return( ST_INVALID );

    case ST_RECORD:
    case ST_PLAY:
        close( dxinfo[ channum ].msg_fd );
        return( ST_GOODBYE );

    case ST_GOODBYE:
    case ST_INVALID:
    case ST_ERROR:
        if (frontend != CT_NTANALOG) {
            set_hkstate( channum, DX_ONHOOK );
            return( ST_WIRING );
        }
    }
}
```

## SCbus Routing Software Reference for Linux

```
    }
    return( ST_ONHOOK );

case ST_ONHOOK:
    return( ST_WIRING );
}
}

/*****
 *      NAME: int process( channum, event )
 * DESCRIPTION: Begin Initial Processing of the Event Received
 *      INPUT: int channum; - Index into dxinfo structure
 *              int event; - Event being processed
 *      OUTPUT: None
 *      RETURNS: New Channel State
 *      CAUTIONS: None
 *****/
int process( channum, event )
    int channum;
    int event;
{
    DX_CST *cstp;
    unsigned short sig;
    short      indx;

    /*
     * Switch according to the event received.
     */
    switch ( event ) {
        case TDX_CST:
            cstp = (DX_CST *) sr_getevtdatap();

            switch ( cstp->cst_event ) {
                case DE_RINGS:
                    /*
                     * Rings Received (Incoming Call)
                     */
                    if ( dxinfo[ channum ].state == ST_WIRING ) {
                        return( ST_OFFHOOK );
                    }
                    break;

                default:
                    break;
            }
            break;

        case TDX_PLAY:      /* Play Completed */
        case TDX_RECORD:   /* Record Completed */
        case TDX_GETDIG:   /* Get Digits Completed */
            /*
             * If drop in loop current, set state to ON-HOOK
             */
            switch ( frontend ) {
                case CT_NTANALOG:
                    if ( ATDX_TERMMSK( dxinfo[ channum ].chdev ) & TM_LCOFF ) {
                        return( ST_ONHOOK );
                    }
                    break;

                case CT_NT11:

```

## 5. Demo Program

```
    if ( ( AIDT_TSSGBIT( dxinfo[ channum ].tsdev ) & DTSG_RCVA ) == 0 ) {
        if ( dxinfo[ channum ].state == ST_WIRING ) {
            return( ST_WIRING );
        } else {
            return( ST_ONHOOK );
        }
    }
    break;

case CT_NTEL:
    if ( ( AIDT_TSSGBIT( dxinfo[ channum ].tsdev ) & DTSG_RCVA ) != 0 ) {
        if ( dxinfo[ channum ].state == ST_WIRING ) {
            return( ST_WIRING );
        } else {
            return( ST_ONHOOK );
        }
    }
    break;
}

case TDX_SETHOOK: /* Set-Hook Complete */
    return( curr_state( channum, event ) );

case DTEV_SIG: /* DTI signalling event */
    sig = (unsigned short)((unsigned short *)sr_getevtdatap());

    for (indx = 0; indx < 4; indx++) {
        /*
         * Check if bit in change mask (upper nibble - lower byte) is set
         * or if this is a WINK (upper nibble - upper byte) event
         */
        if (!(sig & (SIGEVTCHK << indx))) {
            continue;
        }
        switch (sig & (SIGBITCHK << indx)) {
        case DTMM_AON:
            switch ( frontend ) {
            case CT_NTEL:
                if ( dxinfo[ channum ].state == ST_WIRING ) {
                    /*
                     * Rings Received (Incoming Call)
                     */
                    return( ST_OFFHOOK );
                }
                break;

            case CT_NTEL:
                /*
                 * Caller Hangup.
                 */
                return( ST_ONHOOK );
            }
            break;

        case DTMM_AOFF:
            switch ( frontend ) {
            case CT_NTEL:
                if ( dxinfo[ channum ].state == ST_WIRING ) {
                    /*
                     * Rings Received (Incoming Call)
                     */
                    return( ST_OFFHOOK );
                }
            }
        }
    }
}
```

## SCbus Routing Software Reference for Linux

```
    }
    break;

    case CT_NIT1:
        /*
         * Caller Hangup.
         */
        return( ST_ONHOOK );
    }
    break;

    case DIMM_BOFF:
    case DIMM_BON:
    case DIMM_COFF:
    case DIMM_CON:
    case DIMM_DOFF:
    case DIMM_DON:
    case DIMM_WINK:
        break;

    default:
        sprintf( tmpbuff, "Unknown DTEV_SIG Event 0x%hx Received on %s",
                sig, ATDV_NAMEP( dxinfo[ channum ].tsdev ) );
        disp_msg( tmpbuff );
    }
}
break;

default:
    /*
     * Unexpected or Error Termination Event
     */
    sprintf( tmpbuff, "Unknown Event %d, State = %d",
            event, dxinfo[ channum ].state );
    disp_msg( tmpbuff );
    disp_err( channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state );
    return( curr_state( channum, event ) );
}

return( dxinfo[ channum ].state );
}
```

## 5. Demo Program

```
/*
 * NAME: int next_state( channum )
 * DESCRIPTION: Begin the Next State, Initiate the Functions
 * INPUT: int channum; - Index into dxinfo structure
 * OUTPUT: None
 * RETURNS: Pass/Fail Status
 * CAUTIONS: None
 */
int next_state( channum )
{
    int errcode = 0;

    switch ( dxinfo[ channum ].state ) {
    case ST_WTRING:
        /*
         * Channel Waiting for Incoming Call
         */
        disp_status( channum, "Ready to accept a call" );
        break;

    case ST_OFFHOOK:
        /*
         * Call Received, Go Off Hook
         */
        disp_status( channum, "Incoming call" );
        errcode = set_hkstate( channum, DX_OFFHOOK );
        break;

    case ST_INTRO:
        /*
         * Play introduction - intro.vox
         */
        errcode = play( channum, introfd );
        break;

    case ST_GETDIGIT:
        errcode = get_digits( channum, &(dxinfo[ channum ].digbuf) );
        break;

    case ST_RECORD:
        if ( dx_clrldigbuf( dxinfo[ channum ].chdev ) == -1 ) {
            sprintf( tmpbuff, "Cannot clear DTMF Buffer for %s",
                    AITDV_NAMEP( dxinfo[ channum ].chdev ) );
            disp_msg( tmpbuff );
            disp_err( channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state );
        }

        dxinfo[ channum ].msg_fd =
            open( dxinfo[ channum ].msg_name, O_RDWR | O_TRUNC | O_CREAT, 0666 );
        if ( dxinfo[ channum ].msg_fd == -1 ) {
            sprintf( tmpbuff, "Cannot create %s for recording",
                    dxinfo[ channum ].msg_name );
            disp_msg( tmpbuff );
            errcode = -1;
        }

        if ( errcode == 0 ) {
            errcode = record( channum, dxinfo[ channum ].msg_fd );
        }
        break;
    }
}
```

## SCbus Routing Software Reference for Linux

```
case ST_PLAY:
    dxinfo[ channum ].msg_fd = open( dxinfo[channum].msg_name, O_RDWR, 0666 );
    if ( dxinfo[ channum ].msg_fd == -1 ) {
        sprintf( tmpbuff, "Cannot open %s for play-back",
                dxinfo[ channum ].msg_name );
        disp_msg( tmpbuff );
        errcode = -1;
    }

    if ( errcode == 0 ) {
        errcode = play( channum, dxinfo[ channum ].msg_fd );
    }
    break;

case ST_INVALID:
    errcode = play( channum, invalidfd );
    break;

case ST_GOODBYE:
    errcode = play( channum, goodbyefd );
    break;

case ST_ONHOOK:
    errcode = set_hkstate( channum, DX_ONHOOK );
    break;
}

return( errcode );
}

/*****
 *      NAME: void main( argc, argv )
 * DESCRIPTION: Entry Point to Application.
 *      INPUT: int argc; - Argument Count.
 *              char *argv[]; - Array of Pointers to Arguments.
 *      OUTPUT: None.
 *      RETURNS: None.
 *      CAUTIONS: None.
 *****/
void main( argc, argv )
    int argc;
    char *argv[];
{
    int channum; /* Channel Number for Index */
    int chtsdev; /* Channel/Timeslot Device Descriptor */
    int mode;

    /*
     * Process Command Line Arguments
     */
    chkargs( argc, argv );

    /*
     * Set up the Signal Handler
     */
    sigset( SIGHUP, (void (*)()) intr_hdlr );
    sigset( SIGINT, (void (*)()) intr_hdlr );
    sigset( SIGQUIT, (void (*)()) intr_hdlr );
    sigset( SIGTERM, (void (*)()) intr_hdlr );

    /*
```

## 5. Demo Program

```
* Initialize the Display
*/
disp_init();

/*
 * Set the Device to Polled Mode
 */
mode = SR_POLLMODE;
if ( sr_setparm( SRL_DEVICE, SR_MODEID, &mode ) == -1 ) {
    disp_msg( "Unable to set to Polled Mode" );
    QUIT( 1 );
}

/*
 * Initialize System
 */
sysinit();

/**
 ** Main Loop
 **/
while ( 1 ) {
    /*
     * Wait for Completion of an Event
     */
    sr_waitevt( -1 );

    /*
     * Get the channel number from which the event was received.
     */
    channum = 1;
    chtsdev = sr_getevtdev();
    while ( channum <= maxchans ) {
        if ( ( dxinfo[ channum ].chdev == chtsdev ) ||
            ( dxinfo[ channum ].tsdev == chtsdev ) ) {
            break;
        } else {
            channum++;
        }
    }

    if ( channum > maxchans ) {
        sprintf( tmpbuff, "Unknown Event for Device %d - Ignored",
            chtsdev );
        disp_msg( tmpbuff );
        continue;
    }

    /*
     * Process the Event and Save the Next Channel State
     */
    dxinfo[ channum ].state = process( channum , sr_getevttype() );

    /*
     * Begin the New State Checking the Error Code Returned
     */
    if ( next_state( channum ) == -1 ) {
        disp_err( channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state );
    }
}
}
```

## **SCbus Routing Software Reference for Linux**

```
/*
 * If an Error occurs, put the state to ST_ERROR
 * and play the goodbye message and hang-up.
 * If the Play fails then simple, hang-up on the caller.
 */
dxinfo[ channum ].state = ST_ERROR;
if ( play( channum, goodbyefd ) == -1 ) {
    disp_msg( "Unable to play goodbye msg" );
    disp_err(channum, dxinfo[ channum ].chdev, dxinfo[ channum ].state);
    set_hkstate( channum, DX_ONHOOK );
}
}
}
```

## Appendix A

---

### SCbus Routing Function Summary

<b>ag_getctinfo()</b>	returns information about an analog device in a CT_DEVINFO structure
<b>ag_getxmitslot()</b>	returns SCbus time slot information contained in an SC_TSINFO structure that includes the number of the SCbus time slot connected to the analog transmit channel
<b>ag_listen()</b>	connects analog receive (listen) channel to an SCbus time slot
<b>ag_unlisten()</b>	disconnects analog receive (listen) channel from SCbus time slot
<b>dt_getctinfo()</b>	returns information about a digital interface device in a CT_DEVINFO structure
<b>dt_getxmitslot()</b>	returns SCbus time slot information contained in an SC_TSINFO structure that includes the number of the SCbus time slot connected to the digital transmit channel
<b>dt_listen()</b>	connects digital receive (listen) channel to an SCbus time slot
<b>dt_unlisten()</b>	disconnects digital receive (listen) channel from SCbus time slot
<b>dx_getctinfo()</b>	returns information about a voice device in a CT_DEVINFO structure
<b>dx_getxmitslot()</b>	returns SCbus time slot information contained in an SC_TSINFO structure that includes the number of the SCbus time slot connected to the voice transmit channel
<b>dx_listen()</b>	connects voice receive (listen) channel to an SCbus time slot
<b>dx_unlisten()</b>	disconnects voice receive (listen) channel from SCbus time slot
<b>fx_getxmitslot()</b>	returns SCbus time slot information contained in an SC_TSINFO structure that includes the number of the SCbus time slot connected to the fax transmit channel
<b>fx_listen()</b>	connects fax receive (listen) channel to an SCbus time slot
<b>fx_unlisten()</b>	disconnects fax receive (listen) channel from SCbus time slot
<b>nr_scroute()</b>	makes half or full-duplex connection between two SCbus devices
<b>nr_scunroute()</b>	breaks half or full-duplex connection between two SCbus devices

*SCbus Routing Software Reference for Linux*

## Appendix B

---

### Related Publications

This appendix lists publications you should refer to for additional information on Dialogic products and SCbus technology.

- *SCbus Routing Software Reference for Unix*
- *Voice Software Reference for Linux*
- *Voice Software Installation Reference for Linux*
- *Digital Network Interface Software Reference*
- *Fax Software Reference for Linux*

*SCbus Routing Software Reference for Linux*

# Index

---

—

\_getctinfo(), 2

\_getxmitslot(), 2

\_listen(), 2

\_unlisten(), 2

## A

ag\_, 1

ag\_getctinfo(), 3, 18

ag\_getxmitslot(), 3, 22

ag\_listen(), 3, 26

ag\_unlisten(), 3, 30

analog signaling, 26

## C

convenience function, 6, 12

ct\_busencoding field, 19, 34, 48, 75

ct\_busmode, 19, 34, 48, 74

ct\_devfamily, 19, 34, 48, 74

CT\_DEVINFO structure, 73

ct\_devmode field, 19, 34, 48, 74

ct\_nettype field, 19, 34, 48, 74

ct\_prodid field, 19, 33, 48, 74

## D

data structures, 73

demo program, 77

dt\_, 2

dt\_getctinfo(), 3, 33

dt\_getxmitslot(), 3, 37

dt\_listen(), 3, 40

dt\_unlisten(), 3, 44

DTI functionality, 12

DTISC define, 12

dx\_, 2

dx\_getctinfo(), 3, 47

dx\_getxmitslot(), 3, 51

dx\_listen(), 3, 54

dx\_unlisten(), 3, 58

## F

fax functionality, 12

FAXSC define, 12

full-duplex, 54, 64

full-duplex connection, 6, 12, 26

function reference, 17

fx\_, 2

fx\_getxmitslot(), 4, 61

fx\_listen(), 4, 64

fx\_unlisten(), 4, 69

## H

half-duplex, 54, 64

half-duplex connection, 6, 12, 26

## **L**

Linux demo program, 77  
answering machine demo, 77

## **N**

nr\_sc prefix, 12  
nr\_scroute(), 1, 6  
nr\_scunroute(), 1, 12

## **P**

pansr.c, 77

## **R**

related publications, 99

## **S**

sample program, 77  
sc\_numts field, 75  
sc\_tsarrayp field, 75  
SC\_TSINFO structure, 73, 75  
Scbus  
ag\_getctinfo(), 3  
ag\_getxmitslot(), 3  
ag\_listen(), 3  
ag\_unlisten(), 3

dt\_getctinfo(), 3  
dt\_getxmitslot(), 3  
dt\_listen(), 3  
dt\_unlisten(), 3  
dx\_getctinfo(), 3  
dx\_getxmitslot(), 3  
dx\_listen(), 3  
dx\_unlisten(), 3  
fx\_getxmitslot(), 4  
fx\_listen(), 4  
fx\_unlisten(), 4

SCbus, 1  
nr\_sc prefix, 12

SCbus convenience functions, 1  
nr\_scroute(), 1, 5  
nr\_unscroute(), 1, 5

SCbus data structures, 73

SCbus routing functions, 1  
overview, 1  
summary, 97

system initialization, 26

## **T**

TDM, 1

## **V**

voice boards  
analog signaling, 26