

Standard Runtime Library Programmer' s Guide for Windows 2000

Copyright ©2001.8.24 SCT Corporation

PRINTED ON RECYCLED PAPER

05-0168-009

Table of Contents

1. SRL Overview.....	5
1.1. About This SRL Guide	5
1.2. SRL Product Terminology.....	6
1.3. SRL Description	6
1.4. SRL Event Management Functions	7
1.5. SRL Standard Attribute Functions.....	8
1.6. Application Context Management	8
1.7. SRL Termination Parameter Table	8
2. Basic SRL Programming Concepts for SCT Products.....	9
2.1. SRL Devices.....	9
2.2. SRL Programming Options	9
2.3. SRL Synchronous Programming	9
2.4. SRL Asynchronous Programming	10
2.5. SRL Extended Asynchronous Programming	10
3. Basic SRL Programming Models	11
3.1. Selecting Basic SRL Programming Models.....	11
3.2. Synchronous Model.....	11
Synchronous Model Advantages	
Synchronous Model Disadvantages	
Synchronous Model Programming Notes	
Synchronous Model Example	
3.3. Asynchronous Model.....	15
Asynchronous Model Advantages	
Asynchronous Model Disadvantages	
Asynchronous Model Programming Notes	
3.4. Extended Asynchronous Model.....	16
Extended Asynchronous Model Advantages	
Extended Asynchronous Model Disadvantages	
Extended Asynchronous Model Programming Notes	
Extended Asynchronous Model Example	
4. Advanced SRL Programming Concepts for SCT Products.....	24
4.1. SRL Event Handler Programming.....	24
4.1.1. Setting Up Event Handlers	

4.1.2. Event Handler Guidelines	
4.1.3. Hierarchy of Event Handlers	
4.1.4. SRL Callback Thread Behavior	
4.2. Asynchronous with Windows Callback Programming	27
4.3. Asynchronous with Win32 Synchronization Programming.....	28
5. Advanced SRL Programming Models	29
5.1. Selecting Advanced SRL Programming Models	29
5.2. Synchronous with SRL Callback Model.....	30
Synchronous with SRL Callback Model Advantages	
Synchronous with SRL Callback Model Disadvantages	
Synchronous with SRL Callback Model Programming Notes	
Synchronous with SRL Callback Model Example	
5.3. Asynchronous with SRL Callback Model.....	40
Asynchronous with SRL Callback Model Advantages	
Asynchronous with SRL Callback Model Disadvantages	
Asynchronous with SRL Callback Model Programming Notes	
Asynchronous with SRL Callback Model Example	
5.4. Asynchronous with Windows Callback Model.....	46
Asynchronous with Windows Callback Model Advantages	
Asynchronous with Windows Callback Model Disadvantages	
synchronous with Windows Callback Model Programming Note	
Asynchronous with Windows Callback Model Example	
5.5. Asynchronous with Win32 Synchronization Model	55
Asynchronous with Win32 Synchronization Model Advantages	
Asynchronous with Win32 Synchronization Model Programming Notes	
Asynchronous with Win32 Synchronization Model Example	
6. SRL Event Management Functions	63
6.1. Event Management Function Overview.....	63
6.1.1. Event Handling Functions	
6.1.2. Event Data Retrieval Functions	
6.1.3. SRL Parameter Functions	
6.2. SRL Error Handling	64
6.3. SRL Event Management Functions Include Files.....	65
6.4. SRL Event Management Function Reference Overview	65
sr_dishdlr() - disables the handler	
sr_enbhdlr() - enables the handler function	
sr_getboardcnt() - retrieves the number of boards of a particular type	
sr_GetDllVersion() - returns the SRL DLL Version Number	

sr_getevtdatap() - returns the address of the variable data block

sr_getevtdev() - returns the SCT device handle

sr_getevtlen() - returns the length of the variable data

sr_libinit() - initializes the Standard Runtime Library DLL

sr_NotifyEvent() - send event notification to a window

sr_putevt() - Add an event to the SRL event queue

sr_waitevt() - wait for any event to occur

sr_waitevtEx() - waits for events on certain devices

7. SRL Standard Attribute Functions..... 99

7.1. SRL Standard Attribute Functions Overview 99

7.2. SRL Standard Attribute Functions Include Files 99

7.3. SRL Standard Attribute Function Reference Overview..... 99

ATDV_IRQNUM() -

ATDV_NAMEP() - returns a pointer to an ASCII string

ATDV_SUBDEVS() - returns the number of subdevices for the device

8. DV_TPT Termination Parameter Table Structure110

DV_TPT structures

Description of the typedef for the DV_TP

Glossary..... 112

1. SRL Overview

- SRL (Section 1.1).
- SRL (Section 1.2).
- SRL (Section 1.3).
- SRL event management function (Section 1.4).
- SRL standard attribute function (Section 1.5).
- SRL Termination parameter (Section 1.6).

1.1 About This SRL Guide

software SCT device windows 2000

windows 2000 SCT SRL

- event management functions
- Standard attribute functions

1 SRL Overview windows 2000 SCT SRL

2 SCT product SRL

3 SRL

-
-
-

4 . SCT products SRL SRL

5 . SRL

- Synchrononous with SRL Callback
- Asynchronous with SRL Callback

- Asynchronous with Windows Callback
- Asynchronous with Win32 Synchronization

6 . SRL event management function event management function ,
 , , .

7 . SRL attribute standard attribute function ,
 , .

8 . DV_TPT Termination Parameter Table structure SRL data structure
 DV_TPT .

C
 가 가 .

1.2. SRL Product Terminology

guide .

VPMxxx SCT series .

SRL windows 2000 SCT standard Runtime library .
 windows 2000 SCT Voice software SCT
 Standard Runtime Library .

1.3. SRL Description

SRL event handling common interface SCT Device
 . SRL SCT board event
 . SRL , event
 .

SCT SRL application C data structure Library
 .

Event management C functions
 Standard attribute C functions
 Application

management

A termination parameter table data structure

SRL

Srllib.h

Libsrlmt.lib

Libsrlmt.dll

Srllib.c

Srllib.cpp

Multi-threaded library

SRL

가

3

SRL

5

SRL

application

SRL

windows

performance

Tight integration with the Windows programming model

New options for program development

1.4. SRL Event Management Functions

event management function device event interface

application (,) program flow

application

return , event . EV_ASYNC

가

operation

가

block . EV_SYNC

device event

SRL Callback Programming model

device

application-defined event handler

3 SRL

5 SRL

application

6 SRL Event Management Function

1.5. SRL Standard Attribute Functions

Standard attribute device name, board type device library

NOTE: SRL SRL_DEVICE ,device attribute 가
 , event .

7 SRL Standard Attribute Function standard attribute function

1.6. Application Context Management

Application Context Management application set-up device device
 context . User context 2가 :

- Device application table index
- Device application structure pointer

sr_setparm() user-specific context sr_getparm()
 user-specific context .

1.7. SRL Termination Parameter Table

SRL TPT . TPT srllib.h file SCT device
termination DV_TPT dadta structure
 . 8 DV_TPT Termination Parameter Table structure DV_TPT structure

2. basic SRL Programming Concepts for SCT Products

- SRL Device (Section 2.1).
- SRL Programming option (Section 2.2).
- SRL synchronous programming (Section 2.3).
- SRL asynchronous programming (Section 2.4).
- SRL extended asynchronous programming (Section 2.5).

2.1. SRL Devices

SRL device SRL address 가 entity.

- Voice channels
- Digital time slots
- FAX devices
- SPM station sets
- Board devices

2.2. SRL Programming Options

SRL windows 2000

SRL windows 2000

- , programming
- SCT event handler
- programmer가 include operation
- programming
- programming
-
- work thread event handling

2.3. SRL Synchronous Programming

operating system SCT Device thread가 action
 , device thread sleep 가
 , operating system sleep thread . ,
 vpm_play() application file play ,
 thread play가 block , play가 thread가

3 Basic SRL programming model

2.4. SRL Asynchronous Programming

programming , calling thread SCT 가
 operation . 가 , application 가
 event . programming application multitasking
 , device .
 single device
 . SCT device
 . SRL device event driven state machine
 . SCT 가 block ,
 , thread . event가 SRL ,
 SCT , state machine .

non-SCT application state single -point state processing
 SCT event queue event .

- , :
- single SCT device 가 .
 - 가 application .
 - application porting extended mechanism .
 - Win32 API MFC windows 32-bit SRL
 SRL .
 - system overhead context .
 - SCT device event .

2.5. SRL Extended Asynchronous Programming

, device
 .
 application device
 state machine , application thread process space
 database resource , fax service group
 service group 가 .

3. Basic SRL Programming Models

SRL programming model

- Synchronous
- Asynchronous
- Extended Asynchronous

3.1. Selecting Basic SRL Programming Models

Basic Programming model application

Basic Programming model

Table 1. Guidelines for Selecting a Basic Programming Model

Application Requirements	Recommended Basic Programming Model	Threading and Event Handling Considerations
Few devices	Synchronous (Section 3.2)	SCT Device thread
Many devices Multiple tasks	Asynchronous (Section 3.3)	Event sr_waitevt() SCT Device thread
Many devices Multiple tasks Grouping device가 , group event 가	Extended Asynchronous (Section 3.4)	Grouping device , event sr_waitevtEx()

3.2. Synchronous Model

가 programming , Voice-processing device code , code SCT Device thread . SCT function , event_driven state machine application

- device가
- device action flow control

Synchronous Model Advantages

가

Synchronous Model Disadvantages

Main 가 SCT device

. device
scalability .

SCT operation thread blocks processing

가 vpm_getevt() SCT function event

Synchronous Model Programming Notes

Program flow가 , action ,
flow control .
block , application
SCT device .

Synchronous Model Example

programming

```
/* C includes */
#include <stdio.h>
#include <process.h>
#include <conio.h>
#include <ctype.h>
#include <errno.h>
#include <windows.h>
/* SCT includes */
#include <srllib.h>
#include <dxxplib.h>
/* Defines */
#define MAX_CHAN 4 /* maximun number of voice channels in system */
/* Globals */
```

```

int Thrd_num[MAX_CHAN];

int Kbhit_flag = 0;

/* Prototypes */

int main();

DWORD WINAPI sample_begin(LPVOID);

/*****

* NAME : int main()

* DESCRIPTION : prepare screen for ouput, create threads and

* : poll for keyboard input

* INPUT : none

* OUTPUT : none

* RETURNS : 0 on success; 1 if a failure was encountered

* CAUTIONS : none

*****/

int main()
{
    int cnt;

    HANDLE thread_handles[MAX_CHAN];

    DWORD threadID;

    /* show application's title */

    printf("Synchronous Mode Sample Application - hit any key to exit...\n");

    /* create one thread for each voice channel in system */
    for (cnt = 0; cnt < MAX_CHAN; cnt++) {
        Thrd_num[cnt] = cnt;

        if ((thread_handles[cnt] = (HANDLE)_beginthreadex(NULL,0,sample_begin,
            (LPVOID)&Thrd_num[cnt],0,&threadID)) == (HANDLE)-1) {
            printf("ERROR: Could not create thread #d: errno = %d\n", cnt,
                errno);

            exit(1);
        }
    }

    /* wait for Keyboard input to shutdown program */
    getch();

    Kbhit_flag++; /* let threads know it's time to abort */

    /* sleep here until all threads have completed their tasks */
    if (WaitForMultipleObjects(MAX_CHAN, thread_handles, TRUE, INFINITE)
        == WAIT_FAILED) {

```

```

        printf("ERROR: Failed WaitForMultipleObjects(): error = %ld\n",
               GetLastError());
    }
    return(0);
}

/*****
* NAME : DWORD WINAPI sample_begin(LPVOID argp)
* DESCRIPTION : do all channel specific processing
* INPUT : LPVOID argp - pointer to the thread's index number
* OUTPUT : none
* RETURNS : 0 on success; 1 if a failure was encountered
* CAUTIONS : none
*****/
DWORD WINAPI sample_begin(LPVOID argp)
{
    char channame[20];
    int chdesc;
    int thrd_num = *((int *)argp);
    /* build name of voice channel */
    sprintf(channame, "vpmB%dC%d", (thrd_num / 4) + 1,
            (thrd_num % 4) + 1);
    /* open voice channel */
    if ((chdesc = vpm_open(channame, 0)) == -1) {
        printf("%s - FAILED: vpm_open(): errno = %d\n",
               channame, vpm_fileerrno());
        return(1);
    }
    printf("%s - Voice channel opened\n", ATDV_NAMEP(chdesc));
    /* loop until Keyboard input is received */
    while (!Kbhit_flag) {
        /* set the voice channel off-hook */
        if (vpm_sethook(chdesc, DX_OFFHOOK, EV_SYNC) == -1) {
            printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
                   ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
            return(1);
        }
    }
    printf("%s - Voice channel off-hook\n", ATDV_NAMEP(chdesc));
    /* dial number (without call progress) */

```

```

printf("%s - Voice channel dialing...\n",ATDV_NAMEP(chdesc));
if (vpm_dial(chdesc, "12025551212", NULL, EV_SYNC) == -1) {
    printf("%s - FAILED: vpm_dial(): %s (error #%d)\n",
        ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
    return(1);
}

printf("%s - Voice channel Done dialing\n",ATDV_NAMEP(chdesc));
/* set the voice channel back on-hook */
if (vpm_sethook(chdesc, DX_ONHOOK, EV_SYNC) == -1) {
    printf("%s - FAILED: vpm_sethook(DX_ONHOOK): %s (error #%d)\n",
        ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
    return(1);
}

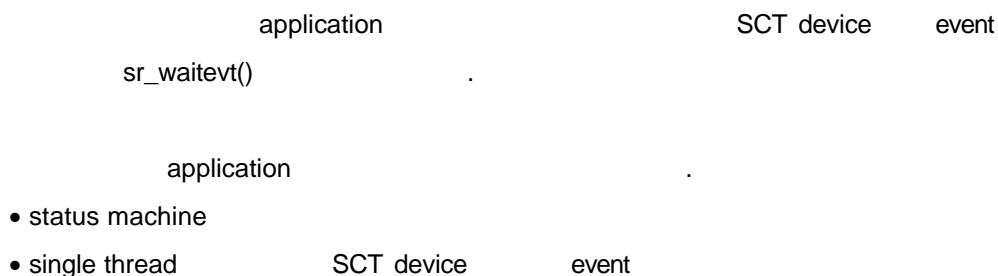
printf("%s - Voice channel on-hook\n",ATDV_NAMEP(chdesc));
}

/* close the voice channel */
if (vpm_close(chdesc) == -1) {
    printf("%s - FAILED: vpm_close(): %s (error #%d)\n",
        ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
    return(1);
}

printf("%s - Voice channel closed\n",channname);
return(0);
}

```

3.3. asynchronous Model



Asynchronous Model Advantages

- SCT device thread system
- application SCT

Asynchronous Modes Disadvantages

- application state machine

Asynchronous Model Programming Notes

- event가 event access

sr_getevtddev() event가 SCT Device handle

sr_getevtttype() event eventtype

sr_getevtdatap() event 가 pointer

sr_getevtlen() sr_getevtdatap() 가 byte

- event-specific data **sr_getevtdatap()** event
sr_waitevt()가

- event가 , application
event가 , event가

3.4. Extended Asynchronous Model

application thread 가 device group

sw_waitevtEx()

application Extended Asynchronous model

- state machine
- device group event

Extended Asynchronous Model Advantages

- SCT device
- model application SCT thread

Extended Asynchronous Modes Disadvantages

- application state application .

Extended Asynchronous Model Programming Notes

- , **sr_waitevtEx()** .
sr_waitevt() .
- event가 , event [access

sr_getevtdev() event가 SCT Device handle

sr_getevttype() event eventtype

sr_getevtdatap() event 가 pointer .

sr_getevtlen() sr_getevtdatap() 가 byte .

- event-specific data **sr_getevtdatap()** . event
sr_waitevt()가

- event가 , application
event가 , event가

- SCT Device group . ,
thread가 event 가 .
- **sr_waitevtEx()** **sr_waitevt()** event handler

Extended Asynchronous Model Example

Extended Asynchronous model 가 .

```
*****
* NAME : int main()
* DESCRIPTION : create thread and poll for keyboard input
* INPUT : none
* OUTPUT : none
* RETURNS : 0 on success; 1 if a failure was encountered
* CAUTIONS : none
```

```

*****/

int main()
{
    HANDLE thread_handle[2];

    DWORD threadID;

    /* show application's title */
    printf("Extended Asynchronous Mode Sample Application - hit any key
                                                to exit...\n");

    /* create one thread to run one state machine */
    if ((thread_handle[0] = (HANDLE)_beginthreadex(NULL,
0,
StateMachine1,
(LPVOID)0,
0,
&threadID)) == (HANDLE)-1) {
        printf("ERROR: Could not create thread : errno = %d\n", errno);
        exit(1);
    }

    /* create a second thread to run the other state machine */
    if ((thread_handle[1] = (HANDLE)_beginthreadex(NULL,
0,
StateMachine2,
(LPVOID)2,
0,
&threadID)) == (HANDLE)-1) {
        printf("ERROR: Could not create thread : errno = %d\n", errno);
        exit(1);
    }

    /* wait for Keyboard input to shutdown program */
    getch();

    Kbhit_flag++; /* let thread know it's time to abort */

    /* sleep here until thread has terminated */
    if (WaitForMultipleObjects(2, thread_handle, TRUE, INFINITE)
        == WAIT_FAILED) {
        printf("ERROR: Failed WaitForMultipleObjects(): error = %ld\n",
            GetLastError());
    }

    return(0);
}

```

```

}

/*****

* NAME : DWORD WINAPI StateMachine1(LPVOID argp)
* DESCRIPTION : This tread runs the offhook-dial-onhook state machine
* INPUT : LPVOID argp - NULL pointer (not used)
* OUTPUT : none
* RETURNS : 0 on success; 1 if a failure was encountered
* CAUTIONS : none

*****/

DWORD WINAPI StateMachine1(LPVOID argp)
{
    char channame[20];
    int chdesc;
    int cnt;
    int hDevice[MAX_CHAN];
    int hEvent;
    long EventCode;
    int basechn = (int)argp;
    for (cnt = basechn; cnt < basechn + (MAX_CHAN/2); cnt++) {
        /* build name of voice channel */
        sprintf(channame, "vpmB%dC%d", (cnt / 4) + 1, (cnt % 4) + 1);
        /* open voice channel */
        if ((chdesc = vpm_open(channame, 0)) == -1) {
            printf("%s - FAILED: vpm_open(): errno = %d\n",
                channame, vpm_fileerrno());
            return(1);
        }
        hDevice[cnt] = chdesc;
        printf("%s - Voice channel opened\n", ATDV_NAMEP(chdesc));
        /* kick off the state machine by going offhook asynchronously */
        if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
            printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
                ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
            return(1);
        }
        printf("%s - Voice channel off-hook initialized\n", ATDV_NAMEP(chdesc));
    }
}

/* loop until Keyboard input is received */

```

```

while (!Kbhit_flag) {
    /*
    * wait for event on the specific list of handles
    */
    sr_waitevtEx(&hDevice[basechn], MAX_CHAN/2, -1, &hEvent);
    /*
    * gather data about the event
    */
    chdesc = sr_getevtdev(hEvent);
    EventCode = sr_getevtttype(hEvent);
    switch(EventCode) {
    case TDX_SETHOOK:
        if (VPMX_HOOKST(chdesc) == DX_OFFHOOK) {
            printf("%s - Voice channel off-hook\n", ATDV_NAMEP(chdesc));
            /* we went off hook so start dialing */
            if (vpm_dial(chdesc, "12025551212", NULL, EV_ASYNC) == -1) {
                printf("%s - FAILED: vpm_dial(): %s (error #d)\n",
                    ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc),
                    ATDV_LASTERR(chdesc))
                ;
                return(1);
            }
            printf("%s - Voice channel dialing initialized\n",
                ATDV_NAMEP(chdesc));
        } else {
            /* we went on hook so go off hook again */
            printf("%s - Voice channel on-hook\n", ATDV_NAMEP(chdesc));
            /* set the voice channel off-hook */
            if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
                printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error
                #d)\n",
                    ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
                return(1);
            }
            printf("%s - Voice channel off-hook
            initialized\n", ATDV_NAMEP(chdesc));
        }
        break;
    }
}

```

```

        case TDX_DIAL:
            printf("%s - Voice channel Done dialing\n",ATDV_NAMEP(chdesc));
            /* done dialing so set the voice channel on-hook */
            if (vpm_sethook(chdesc, DX_ONHOOK, EV_ASYNC) == -1) {
                printf("%s - FAILED: vpm_sethook(DX_ONHOOK): %s (error #%d)\n",
                    ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
                return(1);
            }
            printf("%s - Voice channel on-hook initialized\n",ATDV_NAMEP(chdesc));
            break;

        default:
            printf("Received unexpected event 0x%X on device %d\n", EventCode,
                chdesc);

            break;
    }
}

for (cnt = basechn; cnt < basechn + (MAX_CHAN/2); cnt++) {
    /* close the voice channel */
    chdesc = hDevice[cnt];
    printf("%s - Voice channel closing\n",ATDV_NAMEP(chdesc));
    if (vpm_close(chdesc) == -1) {
        printf("%s - FAILED: vpm_close(): %s (error #%d)\n",
            ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
        return(1);
    }
}

return(0);
}

/*****
* NAME : DWORD WINAPI StateMachine2(LPVOID argp)
* DESCRIPTION : This tread runs the offhook-playtone-onhook state machine
* INPUT : LPVOID argp - NULL pointer (not used)
* OUTPUT : none
* RETURNS : 0 on success; 1 if a failure was encountered
* CAUTIONS : none
*****/

DWORD WINAPI StateMachine2(LPVOID argp)
{

```

```

char channame[20];
int chdesc;
int cnt;
int hDevice[MAX_CHAN];
int hEvent;
long EventCode;
int basechn = (int)argp;
TN_GEN ToneGeneration;
for (cnt = basechn; cnt < basechn + (MAX_CHAN/2); cnt++) {
/* build name of voice channel */
sprintf(channame, "vpmB%dC%d", (cnt / 4) + 1,
(cnt % 4) + 1);
/* open voice channel */
if ((chdesc = vpm_open(channame, 0)) == -1) {
printf("%s - FAILED: vpm_open(): errno = %d\n",
channame, vpm_fileerrno());
return(1);
}
hDevice[cnt] = chdesc;
printf("%s - Voice channel opened\n", ATDV_NAMEP(chdesc));
/* kick off the state machine by going offhook asynchronously */
if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
printf("%s - Voice channel off-hook initialized\n", ATDV_NAMEP(chdesc));
}
/* loop until Keyboard input is received */
while (!Kbhit_flag) {
/*
* wait for event on the specific list of handles
*/
sr_waitevtEx(&hDevice[basechn], MAX_CHAN/2, -1, &hEvent);
/*
* gather data about the event
*/
chdesc = sr_getevtdev(hEvent);

```

```

EventCode = sr_getevtttype(hEvent);
switch(EventCode) {
case TDX_SETHOOK:
if (VPMX_HOOKST(chdesc) == DX_OFFHOOK) {
printf("%s - Voice channel off-hook\n",ATDV_NAMEP(chdesc));
/* we went off hook so build and play the tone */
vpm_bldtngen(&ToneGeneration, 340, 450, -10, -10, 300);
if (vpm_playtone(chdesc, &ToneGeneration, (DV_TPT *)NULL, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_playtone(): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
printf("%s - Voice channel play tone initialized\n",ATDV_NAMEP(chdesc));
} else {
/* we went on hook so go off hook again */
printf("%s - Voice channel on-hook\n",ATDV_NAMEP(chdesc));
/* set the voice channel off-hook */
if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
printf("%s - Voice channel off-hook initialized\n",ATDV_NAMEP(chdesc));
}
break;
case TDX_PLAYTONE:
printf("%s - Voice channel Done playine tone\n",ATDV_NAMEP(chdesc));
/* done playing a tone so set the voice channel on-hook */
if (vpm_sethook(chdesc, DX_ONHOOK, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_sethook(DX_ONHOOK): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
printf("%s - Voice channel on-hook initialized\n",ATDV_NAMEP(chdesc));
break;
default:
printf("Received unexpected event 0x%X on device %d\n", EventCode, chdesc);
break;

```

```

}
}
for (cnt = basechn; cnt < basechn + (MAX_CHAN/2); cnt++) {
/* close the voice channel */
chdesc = hDevice[cnt];
printf("%s - Voice channel closing\n",ATDV_NAMEP(chdesc));
if (vpm_close(chdesc) == -1) {
printf("%s - FAILED: vpm_close(): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
}
return(0);
}

```


4. Advanced SRL Programming Concepts for SCT Products

- 가 .
- SRL handler programming guideline (Section 4.1)
 - windows call back programming 가 asynchronous (Section 4.2)
 - Win32 programming 가 asynchronous (Section 4.3)

4.1 SRL Event Handler Programming

event handler device event SRL .

4.1.1. Setting Up Event Handlers

event , application-level interrupt service routine

event handler .

event handler .

- device event
- device event
- device 가 event , event가 SRL 가 event handler .

programming event handler section .

- 4.1.4. SRL Callback Thread Behavior
- 5.3. Asynchronous with SRL Callback Model

4.1.2. Event Handler Guidelines

- event handler .
- event handler thread가 event SRL handler .
 - thread handler 가 .
 - device event handler .
 - device event handler .
 - handler .
- , sr_enbhdr() SRL thread event handler
- SRL handler thread . SRL handler thread handler가
- .(sr_dishdr())

Handler SRL Handler thread context . main
 thread가 non-SCT main thread device
 hang-up block handler service
 SRL handler context .

SRL Callback 가 SRL Parameter
 SR_MODELTYPE SRL handler thread creation .

4.1.4 SRL Callback thread behavior SRL Callback model 가

application event handler thread .
 • SRL SRL handler thread
 • application-handler thread

SRL handler thread event handler SR_MODELTYPE
 SR_MTASYN setting . Application-handler thread event handler
 SR_MODELTYPE SR_STASYN setting SRL handler thread
 가 .

SR_MODELTYPE , multithread application handler
 application SCT device thread
 . sr_waitvt() thread .
 Handler thread context . application
 scenario SRL handler thread,
 SR_MODELTYPE SR_STASYN .

4.1.3. event handler

SRL device event handler .
 1. device event handler event가
 device .
 2. device event handler devic/event
 specific handler .
 3. event device event, handler
 handler가 . device/event -
 specific handler
 handler가 .

event handler function prototype .
 Long usr_hdlr(unsigned long evhandle);

4.1.4. SRL Callback Thread Behavior

SRL Callback model	application	thread	event
handler	.		

- SRL SRL handler thread
- 가 application handler thread

Using and SRL Handler Thread for SRL Callback

```

graph TD
    subgraph Application_Thread [application thread]
        direction TB
        A1[event handler]
        A2[device]
    end
    subgraph SRL_Handler_Thread [SRL handler thread]
        direction TB
        S1[event handler]
        S2[enbhdr()]
    end
    A1 --> S1
    A2 --> S2

```

[illegible]

```

SRL handler thread  context          event handler          .    event handler
                    sr_waitevt()    SCT function          .

```

State machine event handler . event handler가 1
event . Hierarchy handler .

Using an Application-Handler Thread

```

, program structure
application thread
sr_waitevt()
application-handler
, event handler
. SRL handler
thread
SR_MODELTYPE SR_STASYNCR
thread
.
```

application thread task ,
 sr_waitevt() 가 event . Handler가 non-zero
 sr_waitevt() application thread .

4.2. Asynchronous with Windows Callback Programming

windows	Callback Programming	가		application
window eventing technique		SRL	event	. Windows
Callback Programming	가		application	SCT Device event

가 , SRL message 가 window

Asynchronous with Windows Callback programming:

- window GUI programming technique tight .
 - program system .
 - event single point .
- application ,
- application single thread
- resource management . code inter_thread
- system overhead , device event

4.3. Asynchronous with Win32 Synchronization Programming

Win32 synchronization Win32 synchronization
mechanism SRL event 가
mechanism Reset Event I/O Completion Port . Win32 synchronization
programming application SCT Device event가
SRL user-specific wait point . application event
event process SCT event-retrieval .

- Win32 event synchronization device tight .
- system .
- event processing single point .

programming application program ,
single thread application
resource management . inter-thread
system overhead device event

5. Advanced SRL Programming Models

advanced SRL Programming model .

- Synchronous with SRL Callback
- Asynchronous with SRL Callback
- Asynchronous with Windows Callback
- Asynchronous with Win32 Synchronization

5.1. Selecting Advanced SRL Programming Models

processing application
programming . Table 2
programming . Section table programming

Table 2. Guidelines for Selecting an Advanced Programming Model

Application Requirements	Recommended Advanced Programming Model	Threading and Event Handling Considerations
Few devices Needs to service Unsolicited SCT events	Synchronous with SRL Callback (Section 5.2)	SCT device process thread . Unsolicited SCT event handler event call back 가 sr_enbhdr() .
Many devices Multiple tasks Needs user-defined event handlers	Asynchronous with SRL Callback (Section 5.3)	event handler event Section 4.1.4. SRL callback thread behavior .

Application Requirements	Recommended Advanced Programming Model	Threading and Event Handling Considerations
Many devices Multiple tasks Needs the tightest Possible integration With the Windows Messaging scheme	Asynchronous with Windows Callback (Section 5.4)	User-specify window event 가 가 sr_NotifyEvent() 가 event SCT event sr_waitevt(0)
Many devices Multiple tasks Needs the tightest Possible integration with other Win32 devices	Asynchronous with Win32 Synchronization (Section 5.5)	At Application initialization: · Reset event I.O Completion port Win32 · set up the SRLWIN32INFO structure event Win32 SCT event queue event sr_waitevt()

5.2 Synchronous with SRL Callback Model

SRL Callback Model 가 programming
가 .

SCT device processing
SCT event application .
SCT event callback 가
sr_enbhdr() . SCT event
sr_waitvt() thread .

Synchronous with SRL Callback Model Advantages

- SRL Callback 가 SCT device processing
, unsolicited event SCT event .
- sr_enbhdr() , SRL event handler
SRL handler thread .

Synchronous with SRL Callback Model Disadvantages

- main thread가 SCT device thread system growing
- system limited scalability
- thread event event handler
- thread multi tasking SCT event handler가

Synchronous with SRL Callback Model Programming Notes

- application thread sr_enbhdr() event handler 가 device event type event handler . sr_enbhdr() SRL handler thread
- event handler sr_waitevt() SCT
- 1. , inbound call play record telephony hangup event event handler

Synchronous with SRL Callback Model Example

SRL Callback model 가 Synchronous 가 .

```
#define STRICT
#include <windows.h>
#include <windowsx.h>
#include <afxres.h>
#include <process.h>
#include <dxxlib.h>
#include <srllib.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include "resource.h"
#include <sctools.h>

// Defines

#define MAXCHAN 4 // maximum number of voice channels in system
#define WM_SRNOTIFYEVENT WM_USER + 100
```

```

#define ROWHEIGHT 20

// Modified version of the normal HANDLE_MSG macro in windows.h
#define HANDLE_DLGMSG(hwnd, message, fn) \
case (message): \
return(SetDlgMsgResult(hwnd, uMsg, \
HANDLE_##message((hwnd), (wParam), \
(lParam), (fn))))

// This may be expanded to contain other information such as state typedef struct
vpm_info {
int chdev;
int iter;
} DX_INFO;

// Globals
DX_INFO dxinfo[MAXCHAN+1];
int Kbhit_flag = 0;
char tmpbuf[128];
HANDLE hInst;
char gRowVal[MAXCHAN+1][80];

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
void General_OnCommand(HWND , int , HWND , UINT ); // Window's WM_COMMAND handler
int dlgc_OnCommand(HWND ); // WM_SRNOTIFYEVENT handler
int SCTSysInit(HWND );
void SCTClose(HWND);
int get_ts(int );
void disp_status(HWND, int , char *);

/*****
* NAME : WinMain()
* DESCRIPTION : Windows application entry point
* CAUTIONS : none.
*****/

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hinstPrev, LPSTR lpszCmdLine, int
nCmdShow) {
HWND hWnd;
WNDCLASS wc;
MSG msg;
hInst = hInstance;
if (!hinstPrev)
{

```



```

// Fill in window class structure with parameters that
// describe the main window.
wc.style = CS_HREDRAW | CS_VREDRAW; // Class style(s).
wc.lpfnWndProc = (WNDPROC)WndProc; // Window
Procedure
wc.cbClsExtra = 0; // No per-class
extra data.
wc.cbWndExtra = 0; // No per-window
extra data.
wc.hInstance = hInstance; // Owner of
this class
wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE(IDI_ICON1));
// Icon name from .RC
wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Cursor
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); // Default color
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
wc.lpszClassName = "WinCallBack"; // Name to register
// Register the window class and return success/failure code.
if (!RegisterClass(&wc))
return (FALSE); // Exits if unable to register
}
hWnd = CreateWindowEx(0L, "WinCallBack", "Windows Callback
Demo", WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, 0,
CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
// If window could not be created, return "failure"
if (!hWnd)
return (FALSE);
sr_NotifyEvent(hWnd, WM_SRNOTIFYEVENT, SR_NOTIFY_ON);
ShowWindow(hWnd, SW_SHOW); // Show the window
UpdateWindow(hWnd); // Sends WM_PAINT message
// Get and dispatch messages until a WM_QUIT message is received. while
(GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage(&msg); // Translates virtual key code DispatchMessage(&msg); //
Dispatches message to window
}
return (0);

```

```

}

/*****

* NAME : WndProc()

* DESCRIPTION : Windows Procedure

* CAUTIONS : none.

*****/

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;

    PAINTSTRUCT ps;

    RECT rect;

    int numchan;

    switch (uMsg) {
        // Handle the WM_COMMAND messages HANDLE_MSG(hWnd, WM_COMMAND,
        General_OnCommand);

        case WM_SRNOTIFYEVENT:

            if (dlg_OnCommand( hWnd)) { // SCT event

                SCTClose(hWnd);

                DestroyWindow(hWnd); // if dlg_OnCommand() returns FALSE }

                break;

            case WM_CREATE:

                break;

            case WM_PAINT:

                // get the actual window rectangle

                GetClientRect(hWnd, &rect);

                hDC = BeginPaint(hWnd, &ps);

                // display name of application

                rect.top = ROWHEIGHT;

                sprintf(tmpbuf, "Windows Callback Demo");

                DrawText(hDC, tmpbuf, -1, &rect, DT_SINGLELINE | DT_CENTER);

                // display status of channel

                for (numchan=1; numchan<=MAXCHAN; numchan++) {

                    rect.top = (numchan+2) * ROWHEIGHT;

                    DrawText(hDC, gRowVal[numchan], -1, &rect,

                        DT_SINGLELINE);

                }

                EndPaint(hWnd, &ps);

                break;
    }
}

```

```

case WM_CLOSE:
DestroyWindow(hWnd);
break;

case WM_DESTROY:
PostQuitMessage(0); // Allow GetMessage() to return
FALSE
break;

default:
return (DefWindowProc(hWnd, uMsg, wParam, lParam));
} // switch (uMsg)
}

/*****
* NAME : General_OnCommand()
* DESCRIPTION : Message Handler for WM_COMMAND
* CAUTIONS : none.
*****/

void General_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)
{
switch (id) {
case ID_TEST_EXIT:
DestroyWindow(hWnd); // post WM_DESTROY message for WndProc to exit app break;
case ID_TEST_GO: // create threads here and gray the
"start" menu and
// launch the call control threads
if (SCTSysInit(hWnd)) {
// initialize SCT devices, if error initializing then show error message and exit
MessageBox(hWnd, "Error initializing",
"ERROR", MB_OK | MB_ICONSTOP | MB_APPLMODAL); break;
}
// // ungrey the "stop" menu
EnableMenuItem(GetMenu(hWnd), ID_TEST_STOP, MF_ENABLED );
EnableMenuItem(GetMenu(hWnd), ID_TEST_GO, MF_DISABLED | MF_GRAYED);
break;
case ID_TEST_STOP: // "terminate" the call control
threads and ungray menu items // terminate the call control
threads
SCTClose(hWnd);
// disable the Action/Stop menu item

```

```

EnableMenuItem(GetMenu(hWnd), ID_TEST_STOP, MF_DISABLED | MF_GRAYED);
EnableMenuItem(GetMenu(hWnd),
ID_TEST_GO, MF_ENABLED );
break;
default:
return;
} // switch (id)
}

/*****
* NAME : dlgc_OnCommand()
* DESCRIPTION : Message Handler for WM_SRNOTIFYEVENT
* CAUTIONS : none.
*****/

int dlgc_OnCommand(HWND hWnd)
{
int rc = 0;
int chdev;
int event;
DX_CST *cstp;
static iter=0;
int channum;
if (sr_waitevt(0) == -1) {
sprintf(tmpbuf, "sr_waitevt() ERROR %s", ATDV_ERRMSGP( SRL_DEVICE ));
MessageBox(hWnd,tmpbuf, "ERROR" , MB_OK|MB_APPLMODAL);
return (1);
}
chdev = sr_getevtdev();
event = sr_getevttype();

/*
* Switch according to the event received. */
switch ( event ) {
case TDX_SETHOOK:
cstp = (DX_CST *)sr_getevtdatap(); switch( cstp->cst_event) { case DX_ONHOOK:
/* Go offhook next */
if (vpm_sethook(chdev, DX_OFFHOOK,
EV_ASYNC) == -1) {
sprintf(tmpbuf, "FAILED: vpm_sethook(%s, DX_OFFHOOK): %s (error #%d)", ATDV_NAMEP(chdev),
ATDV_ERRMSGP(chdev),

```

```

ATDV_LASTERR(chdev));
MessageBox(hWnd, tmpbuf,
"ERROR", MB_OK|MB_APPLMODAL); return(1);
}
break;
case DX_OFFHOOK:
/* dial next */
if (vpm_dial(chdev, "12025551212", NULL, EV_ASYNC) == -1) { sprintf(tmpbuf, "FAILED:
vpm_dial(%s): %s (error #%d) ", ATDV_NAMEP(chdev),
ATDV_ERRMSGP(chdev),
ATDV_LASTERR(chdev));
MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL); return(1);
}
break;
}
break;
case TDX_DIAL:
/* Next go onhook */
if (vpm_sethook(chdev, DX_ONHOOK, EV_ASYNC) == -1) {
sprintf(tmpbuf, "FAILED: vpm_sethook(%s, DX_ONHOOK): %s (error #%d) ", ATDV_NAMEP(chdev),
ATDV_ERRMSGP(chdev),
ATDV_LASTERR(chdev));
MessageBox(hWnd, tmpbuf,
"ERROR", MB_OK|MB_APPLMODAL);
return(1);
}
}
sprintf(tmpbuf, "Iteration %d Completed", ++dxinfo[channum].iter); disp_status(hWnd,
channum, tmpbuf );
return (0);
}

/*****
* NAME :
SCTSysInit()
* DESCRIPTION : Initialization of SCT devices
* CAUTIONS : none.
*****/

/ int SCTSysInit(HWND hWnd)

```

```

{
int numchan;
char channame[20];
/* Initial processing for MAXCHANS */
for (numchan=1;numchan<=MAXCHAN;numchan++) {
/* build name of voice channel */
sprintf(channame, "vpmB%dC%d", ((numchan-1) / 4) + 1,
((numchan -1)% 4) + 1);
/* open voice channel */
if ((dxinfo[numchan].chdev = vpm_open(channame, 0)) == -1)
{sprintf(tmpbuf, "FAILED: vpm_open(%s): errno = %d ", channame,
vpm_fileerrno());
MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL);
return(1);
}
/* Start the application by putting the channel in onhook state */
if (vpm_sethook(dxinfo[numchan].chdev, DX_ONHOOK, EV_ASYNC) == -1) {
sprintf(tmpbuf, "FAILED: vpm_sethook(%s): %s (error #%d) ",
ATDV_NAMEP(dxinfo[numchan].chdev), ATDV_ERRMSGP(dxinfo[numchan].chdev),
ATDV_LASTERR(dxinfo[numchan].chdev));
MessageBox(hWnd, tmpbuf,
"ERROR", MB_OK|MB_APPLMODAL); return(1);
}
}
return(0);
}

/*****
* NAME : SCTClose()
* DESCRIPTION : Tier down of SCT devices
* CAUTIONS : none.
*****/

void SCTClose(HWND hWnd)
{
int numchan;
/* Close all voice devices before exiting */
for (numchan=1;numchan<=MAXCHAN;numchan++) {
// attempt to stop the channel vpm_stopch(dxinfo[numchan].chdev, EV_SYNC);
vpm_close(dxinfo[numchan].chdev);
}
}

```

```

}
}
/*****
* NAME: disp_status(hwnd, chnum, stringp)
* INPUTS: chno - channel number (1 - 12)
* stringp - pointer to string to display
* DESCRIPTION: display the current activity on the channel in window 2
* the string pointed to by stringp) using chno as a Y offset
*****/
void disp_status(HWND hwnd, int channum, char *stringp)
{
RECT rect;
// get the entire window rectangle and modify it
GetClientRect(hwnd, &rect);
rect.top = (channum+2) * ROWHEIGHT;
rect.bottom = (channum+3) * ROWHEIGHT;
// buffer the message
sprintf(gRowVal[channum], "Channel %d: %s", channum, stringp);
InvalidateRect(hwnd, &rect, TRUE);
UpdateWindow(hwnd);
}

```

5.3.

5.4. Asynchronous with SRL Callback Model

SRL Callback model 가 Asynchronous application
event handler .

- SRL handler thread
- application-handler thread

, 4.1.4. SRL Callback Thread Behavior and Section .

application SRL callback model 가 Asynchronous

- state machine .
- event handler .

Asynchronous with SRI Callback Model Advantages

- SRL Callback 가 SCT devices thread

resource . Win32 가 system growing
system scalability .
· application non-SCT non-SCT thread block event
handler SCT event . SCT event가
window가 thread scheduling service

Asynchronous with SRL Callback Model Disadvantages

- You might need to set up a way for the event handler to communicate events to another thread.

Asynchronous with SRL Callback Model Programming Notes

4.1.4. SRL Callback Thread Behavior

Asynchronous with SRL Callback Model Example

Main thread SRL Callback model 가 Asynchronous 가

```

/***** Aynchronous with SRL Callback Model - Using the Main Thread *****/
/*
    ● Compiled using Visual C++ 5.0
*/
/* C includes */
#include <windows.h>
#include <stdio.h>
#include <StdLib.H>
#include <process.h>
#include <conio.h>
#include <ctype.h>
#include <errno.h>
#include <String.H>
/* SCT includes */
#include <srllib.h>
#include <dxxplib.h>
/* Defines */
#define MAXCHAN 4 /* maximun number of voice channels in system */
#define FOREVER 1

```



```

#define DIALSTRING "01234"

/* Globals */

int vxh[MAXCHAN];

int errflag = FALSE;

/* Prototypes */

int main();

long fallback_hdlr(unsigned long parm);

int voxinit(void);

int sysexit(int exitcode);

int process_events(void);

/*****

* NAME : int main(void)

* DESCRIPTION : The Entry Point into the application.

* INPUT : none

* OUTPUT : none

* RETURNS : 0 on success; 1 if a failure was encountered

* CAUTIONS : none

*****/

int main(void)
{
    /* Show application's title */
    printf("Asynchronous with Main-Thread Callback Model\n");

    /* Start SCT Devices */
    if (voxinit() == -1) {
        sysexit(-1);
    }

    /* Process events , monitor keyboard input and other activities */
    if (process_events() == -1) {
        sysexit(-1);
    }

    sysexit(0);
    return(0);
}

/*****

* NAME : int voxinit(void)

* DESCRIPTION : Initializes SCT Devices.

* INPUT : none

* OUTPUT : none
*****/

```

```

* RETURNS : 0 on success; -1 if a failure was encountered
* CAUTIONS : none
*****/

int voxinit(void)
{
int index;
int mode;
char devname[32];

/** Set to SR_STASYNC so another thread is not created by the SRL to
** monitor events to pass to the handler. We will use this thread
** to monitor events; creating another thread internally is not
** necessary.
**/
mode = SR_STASYNC;
}

/*
* Set-up the fall-back handler
*/
if (sr_enbhdlr((long)EV_ANYDEV, (unsigned long)EV_ANYEVT, fallback_hdlr) == -1 ) {
printf("ERROR:Unable to set-up the fall back handler \n");
return(-1);
}

/* Open the voice chans now */
for (index = 0; index < MAXCHAN; index++) {
sprintf(devname, "vpmBlC%d", (index+1));
if ((vxh[index] = vpm_open(devname, 0)) == -1) {
printf("ERROR: vpm_open(%s) failed, errno = %d\n",
devname, errno);
return(-1);
}
}

/* Issue the dial without call progres on all the voice chans */
for (index = 0; index < MAXCHAN; index++) {
if (vpm_dial(vxh[index], DIALSTRING, NULL, EV_ASYNC) == -1) {
printf("ERROR: vpm_dial(%s) failed, 0x%X(%s)\n",
ATDV_NAMEP(vxh[index]), ATDV_LASTERR(vxh[index]),
ATDV_ERRMSGP(vxh[index]));
return(-1);
}
}

```

```

}
}
}

/*****
    ● NAME : int process_events(void)
    * DESCRIPTION : The processing loop.
    * INPUT : none
    * OUTPUT : none
    * RETURNS : 0 on success; -1 if a failure was encountered
    * CAUTIONS : none
    *****/

int process_events(void)
{
while (FOREVER) {
if (kbhit() != (int)0) {
return(-1);
}

/* Wait for SCT events 1 second */
sr_waitevt(1000);
if (errflag == TRUE) {
return(-1);
}

/* Do other processing here */
printf("Press Any Key to Exit \n");
}

return(0);
}

/*****
    * NAME : long fallback_hdlr(unsigned long parm)
    * DESCRIPTION : The fallback handler for all SCT events
    * INPUT : parm
    * OUTPUT : return value
    * RETURNS :
    * CAUTIONS : none
    *****/

long fallback_hdlr(unsigned long parm)
{
int index, devh, evttype;

```

```

/* Find out the event received */
devh = sr_getevtdev();
evttype = sr_getevttype();
for (index=0; index<MAXCHAN; index++) {
    if (devh == vxh[index]) {
        break;
    }
}

switch(evttype) {
    case TDX_DIAL :
        printf("%s : Dialing again.\n", ATDV_NAMEP(vxh[index]));
        if (vpm_dial(vxh[index], DIALSTRING, NULL, EV_ASYNC) == -1) {
            printf("ERROR: vpm_dial(%s) failed, 0x%X(%s)\n",
                ATDV_NAMEP(vxh[index]), ATDV_LASTERR(vxh[index]),
                ATDV_ERRMSGP(vxh[index]));
            errflag = TRUE;
        }
        break;

    case TDX_ERROR :
        printf("%s : TDX_ERROR!\n", ATDV_NAMEP(vxh[index]));
        errflag = TRUE;
        break;

    default :
        printf("%s : unknown event(0x%X)\n", ATDV_NAMEP(vxh[index]), evttype);
        errflag = TRUE;
        break;
}

/* Remove the events from the SRL queue */
return(1);
}

/*****
* NAME : void sysexit(exitcode)
* DESCRIPTION : Closes the devices and exits the application.
* INPUT : none
* OUTPUT : none
* RETURNS : exitcode
* CAUTIONS : Exit of the application!
*****/

```

```

int sysexit(int exitcode)
{
    int index;

    /* Close the voice chans now */
    for (index = 0; index < MAXCHAN; index++) {
        if (vpm_close(vxh[index]) == -1) {
            printf("ERROR: vpm_close(%s) failed, 0x%X(%s)\n",
                ATDV_NAMEP(vxh[index]), ATDV_LASTERR(vxh[index]),
                ATDV_ERRMSGP(vxh[index]));
        }
    }
    exit(exitcode);
    return(exitcode);
}

```

5.4 Asynchronous with Windows Callback Model

User-specified window event sr_NotifyEvt()

 . Main application loop standard windows message handling schedule .

1. user-specified message user-specified window .
2. sr_waitevt(0) zero time-out .
3. Event data retrieval function sr_gettevtdev() sr_gettevttype()

 event .
4. .

Window messaging scheme 가 tight 가 application

window callback .

Asynchronous with Windows Callback Model Advantages

- SCT device thread
- system resource . single thread application
- SCT portion .
- window messaging scheme tight .
- window messaging loop device event .

Asynchronous with Windows Callback Model Disadvantages

 loop가 processing window

procedure .

blocking .

Asynchronous with Windows Callback Model Programming Notes

- window message callback .
- window message callback 가 .
 1. sr_NotifyEvent() application thread window message callback 가 .
 2. window message callback application event application SCT event queue zero timeout 가 sr_waitevt() .
 3. event, call
 - sr_gettevtdev() to get the event ' s device handle.
 - sr_gettevttype() to get the event type.
- thread가 non-SCT block callback window thread SCT event . hidden window thread . thread message queue 가 , handle hidden window sr_NotifyEvent() .

Asynchronous with Windows Callback Model Example

Windows Callback model 가 Asynchronous 가 .

```
#define STRICT
#include <windows.h>
#include <windowsx.h>
#include <afxres.h>
#include <process.h>
#include <dxxlib.h>
#include <srllib.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include "resource.h"
#include <sctools.h>

// Defines

#define MAXCHAN 4 // maximum number of voice channels in system
#define WM_SRNOTIFYEVENT WM_USER + 100
```

```

#define ROWHEIGHT 20

// Modified version of the normal HANDLE_MSG macro in windowsx.h
#define HANDLE_DLGMSG(hwnd, message, fn) \
case (message): \
return(SetDlgMsgResult(hwnd, uMsg, \
HANDLE_##message((hwnd), (wParam), \
(lParam), (fn))))

// This may be expanded to contain other information such as state typedef
struct vpm_info {
int chdev;
int iter;
} DX_INFO;

// Globals
DX_INFO dxinfo[MAXCHAN+1];
int Kbhit_flag = 0;
char tmpbuf[128];
HANDLE hInst;
char gRowVal[MAXCHAN+1][80];
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
void General_OnCommand(HWND , int , HWND , UINT ); // Window's WM_COMMAND handler
int dlgc_OnCommand(HWND ); // WM_SRNOTIFYEVENT handler
int SCTSysInit(HWND );
void SCTClose(HWND);
int get_ts(int );
void disp_status(HWND, int , char *);

/*****
* NAME : WinMain()
* DESCRIPTION : Windows application entry point
* CAUTIONS : none.
*****/

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hinstPrev, LPSTR lpszCmdLine, int
nCmdShow)
{
HWND hWnd;
WNDCLASS wc;
MSG msg;
hInst = hInstance;
if (!hinstPrev)

```

```

{
// Fill in window class structure with parameters that
// describe the main window.
wc.style = CS_HREDRAW | CS_VREDRAW; // Class style(s).
wc.lpfnWndProc = (WNDPROC)WndProc; // Window Procedure
wc.cbClsExtra = 0; // No per-class extra data.
wc.cbWndExtra = 0; // No per-window extra data.
wc.hInstance = hInstance; // Owner of this class
wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE(IDI_ICON1)); // Icon
name from .RC
wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Cursor
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); // Default color
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
wc.lpszClassName = "WinCallBack"; // Name to register
// Register the window class and return success/failure code.
if (!RegisterClass(&wc))
return (FALSE); // Exits if unable to register
}

hWnd = CreateWindowEx(0L, "WinCallBack", "Windows Callback Demo",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, CW_USEDEFAULT,
0, NULL, NULL, hInstance, NULL);
// If window could not be created, return "failure"
if (!hWnd)
return (FALSE);

sr_NotifyEvent(hWnd, WM_SRNOTIFYEVENT, SR_NOTIFY_ON);
ShowWindow(hWnd, SW_SHOW); // Show the window
UpdateWindow(hWnd); // Sends WM_PAINT message
// Get and dispatch messages until a WM_QUIT message is received.
while (GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage(&msg); // Translates virtual key code
DispatchMessage(&msg); // Dispatches message to window
}
return (0);
}

/*****
* NAME : WndProc()
* DESCRIPTION : Windows Procedure

```



```

* CAUTIONS : none.

*****/

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;

    PAINTSTRUCT ps;

    RECT rect;

    int numchan;

    switch (uMsg) {
        // Handle the WM_COMMAND messages
        HANDLE_MSG(hWnd, WM_COMMAND, General_OnCommand);

        case WM_SRNOTIFYEVENT:
            if (dlg_OnCommand( hWnd )) { // SCT event
                SCTClose(hWnd);
                DestroyWindow(hWnd); // if dlg_OnCommand() returns 1
            }
            break;

        case WM_CREATE:
            break;

        case WM_PAINT:
            // get the actual window rectangle
            GetClientRect(hWnd, &rect);
            hDC = BeginPaint(hWnd, &ps);
            // display name of application
            rect.top = ROWHEIGHT;
            sprintf(tmpbuf, "Windows Callback Demo");
            DrawText(hDC, tmpbuf, -1, &rect, DT_SINGLELINE|DT_CENTER);
            // display status of channel
            for (numchan=1; numchan<=MAXCHAN; numchan++) {
                rect.top = (numchan+2) * ROWHEIGHT;
                DrawText(hDC, gRowVal[numchan], -1, &rect, DT_SINGLELINE);
            }
            EndPaint(hWnd, &ps);
            break;

        case WM_CLOSE:
            DestroyWindow(hWnd);
            break;

        case WM_DESTROY:

```

```

PostQuitMessage(0); // Allow GetMessage() to return FALSE

break;

default:

return (DefWindowProc(hWnd, uMsg, wParam, lParam));

} // switch (uMsg)

}

/*****

* NAME : General_OnCommand()

* DESCRIPTION : Message Handler for WM_COMMAND

* CAUTIONS : none.

*****/

void General_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)
{
switch (id) {
case ID_TEST_EXIT:
DestroyWindow(hWnd); // post WM_DESTROY message for WndProc to exit
app break;

case ID_TEST_GO: // create threads here and gray the "start" menu
// and launch the call control
threads
if (SCTSysInit(hWnd)) {
// initialize SCT devices, if error initializing then show
error message and exit
MessageBox(hWnd, "Error initializing", "ERROR", MB_OK |
MB_ICONSTOP | MB_APPLMODAL); break;
}

// ungrey the "stop" menu
EnableMenuItem(GetMenu(hWnd), ID_TEST_STOP, MF_ENABLED );
EnableMenuItem(GetMenu(hWnd), ID_TEST_GO, MF_DISABLED | MF_GRAYED);
break;

case ID_TEST_STOP: // "terminate" the call control threads and ungray
menu items
SCTClose(hWnd);

// disable the Action/Stop menu item
EnableMenuItem(GetMenu(hWnd), ID_TEST_STOP, MF_DISABLED | MF_GRAYED);
EnableMenuItem(GetMenu(hWnd), ID_TEST_GO, MF_ENABLED );
break;

default:

```

```

return;
} // switch (id)
}

/*****

* NAME : dlgc_OnCommand()
* DESCRIPTION : Message Handler for WM_SRNOTIFYEVENT
* CAUTIONS : none.

*****/

int dlgc_OnCommand(HWND hWnd)
{
    int rc = 0;
    int chdev;
    int event;
    DX_CST *cstp;
    static iter=0;
    int channum;
    if (sr_waitevt(0) == -1) {
        sprintf(tmpbuf, "sr_waitevt() ERROR %s", ATDV_ERRMSGP( SRL_DEVICE ));
        MessageBox(hWnd,tmpbuf, "ERROR" , MB_OK|MB_APPLMODAL);
        return (1);
    }
    chdev = sr_getevtdev();
    event = sr_getevttype();
    /* Switch according to the event received. */
    switch ( event ) {
    case TDX_SETHOOK:
        cstp = (DX_CST *)sr_getevtdatap();
        switch( cstp->cst_event) {
        case DX_ONHOOK:
            /* Go offhook next */
            if (vpm_sethook(chdev, DX_OFFHOOK, EV_ASYNC) == -1) {
                sprintf(tmpbuf,"FAILED: vpm_sethook(%s, DX_OFFHOOK): %s (error
                %d)", ATDV_NAMEP(chdev), ATDV_ERRMSGP(chdev), ATDV_LASTERR(chdev));
                MessageBox(hWnd, tmpbuf, "ERROR",MB_OK|MB_APPLMODAL);
                return(1);
            }
        }
        break;
    case DX_OFFHOOK:

```

```

/* dial next */
if (vpm_dial(chdev, "12025551212", NULL, EV_ASYNC) == -1) {
    sprintf(tmpbuf, "FAILED: vpm_dial(%s): %s (error #%d) ",
        ATDV_NAMEP(chdev), ATDV_ERRMSGP(chdev), ATDV_LASTERR(chdev));
    MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL);
    return(1);
}
break;
}
break;
case TDX_DIAL:
    /* Next go onhook */
    if (vpm_sethook(chdev, DX_ONHOOK, EV_ASYNC) == -1) {
        sprintf(tmpbuf, "FAILED: vpm_sethook(%s, DX_ONHOOK): %s (error #%d) ",
            ATDV_NAMEP(chdev), ATDV_ERRMSGP(chdev), ATDV_LASTERR(chdev));
        MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL);
        return(1);
    }
    break;
}
sprintf(tmpbuf, "Iteration %d Completed", ++dxinfo[channum].iter);
disp_status(hWnd, channum, tmpbuf );
return (0);
}

/*****
* NAME : SCTSysInit()
*   DESCRIPTION : Initialization of SCT devices
* CAUTIONS : none.
*****/

int SCTSysInit(HWND hWnd)
{
    int numchan;
    char channame[20];

    /* Initial processing for MAXCHANS */
    for (numchan=1; numchan<=MAXCHAN; numchan++) {
        /* build name of voice channel */
        sprintf(channame, "vpmB%dC%d", ((numchan-1) / 4) + 1,
            ((numchan -1)% 4) + 1);

```

```

/* open voice channel */
if ((dxinfo[numchan].chdev = vpm_open(channname, 0)) == -1) {
    sprintf(tmpbuf, "FAILED: vpm_open(%s): errno = %d ", channname,
    vpm_fileerrno());
    MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL);
    return(1);
}

/* set user specific information in the device, in this
** case the channel number
*/

/* Start the application by putting the channel in onhook state */
if (vpm_sethook(dxinfo[numchan].chdev, DX_ONHOOK, EV_ASYNC) == -1) {
    sprintf(tmpbuf, "FAILED: vpm_sethook(%s): %s (error #%d) ",
    ATDV_NAMEP(dxinfo[numchan].chdev), ATDV_ERRMSGP(dxinfo[numchan].chdev),
    ATDV_LASTERR(dxinfo[numchan].chdev));
    MessageBox(hWnd, tmpbuf, "ERROR", MB_OK|MB_APPLMODAL); return(1);
}
}

return(0);
}

/*****
* NAME : SCTClose()
* DESCRIPTION : Tier down of SCT devices
* CAUTIONS : none.
*****/

void SCTClose(HWND hWnd)
{
    int numchan;

    /* Close all voice devices before exiting */
    for (numchan=1; numchan<=MAXCHAN; numchan++) {
        // attempt to stop the channel vpm_stopch(dxinfo[numchan].chdev, EV_SYNC);
        vpm_close(dxinfo[numchan].chdev);
    }
}

/*****
* NAME: disp_status(hWnd, chnum, stringp)
* INPUTS: chno - channel number (1 - 12)
* stringp - pointer to string to display
*****/

```

```

* DESCRIPTION: display the current activity on the channel in window 2
* (the string pointed to by stringp) using chno as a Y offset
*****/
void disp_status(HWND hWnd, int channum, char *stringp)
{
RECT rect;
// get the entire window rectangle and modify it
GetClientRect(hWnd, &rect);
rect.top = (channum+2) * ROWHEIGHT;
rect.bottom = (channum+3) * ROWHEIGHT;
// buffer the message
sprintf(gRowVal[channum], "Channel %d: %s", channum, stringp);
InvalidateRect(hWnd, &rect, TRUE);
UpdateWindow(hWnd);
}

```

5.5. Asynchronous with Win32 Synchronization Model

Win32 synchronization 가 Win32 synchronization

SRL event application .

Reset Event I/O completion port . application SCT device

event가 Reset Event I/O completion point

SRL .

Application event process SCT event

retrieval .

Win32 device 가 가 tight application Win32

synchronization 가 .

Asynchronous with Win32 Synchronization Model Advantages

- Win32 synchronization SCT device thread
- Win32 synchronization growing system scalability
- DM3 device source device single point event
- device가 single thread control inter-thread

communication .

· multi-processing platform level scalability . I/O
completion port .

Asynchronous with Win32 Synchronization Model Programming Notes

Reset event application .

Application

1. event Win32 CreateEvent() .
2. reset event가 SRL WIN32INFO structure
. DWHANDLETYPE field SR_RESETEVENT .
ObjectHandle field CreateEvent() event handler .

To wait for event notification:

1. Win32 WaitForSingleObject() WaitForMultipleObjects() .
WaitForMultipleObjects() application source event
가 .
2. , event source가 SCT device SCT event queue
event zero timeout 가 sr_waitevt() .

I/O Completion port application .

Application :

1. completion port Win32 CreateIoCompletionPort() .
2. I/O Completion point가 SRLWIN32INFO
. DwhandleType field SR_IOCOMPLETIONPORT .
ObjectHandle field CreateIoCompletionPort()
completion port . DwhandleType field event
user-specified key . LpOverlapped field optional user -
specified OVERLAPPED structure .

To wait for event notification:

1. SRL device source event Win32
GetQueuedCompletionStatus() .
2. , event source가 SCT device SCT event queue
event zero timeout 가 sr_waitevt() .

Asynchronous with Win32 Synchronization Model Example

An example of the Asynchronous with Win32 Synchronization model is shown below:

```
/* C includes */
#include <stdio.h>
#include <process.h>
#include <conio.h>
#include <ctype.h>
#include <errno.h>
#include <windows.h>
/* SCT includes */
#include <srllib.h>
#include <dxxplib.h>
/* Defines */
#define MAX_CHAN 4 /* maximun number of voice channels in system */
#define SCT_KEY 0 /* Index for SCT reset event */
#define KEYBOARD_KEY 1 /* Index for keyboard reset event */
#define MAX_RESET_EVENTS 2 /* number of reset events */
/* Globals */
int Kbhit_flag = 0;
HANDLE hEvent[MAX_RESET_EVENTS];
/* Prototypes */
int main();
DWORD WINAPI sample_begin(LPVOID);
/*****
* NAME : int main()
* DESCRIPTION : create reset events, create thread and
* : poll for keyboard input
* INPUT : none
* OUTPUT : none
* RETURNS : 0 on success; 1 if a failure was encountered
* CAUTIONS : none
*****/
int main()
{
HANDLE thread_handle;
```



```

DWORD threadID;

DWORD Index;

/* show application's title */

printf("Async with Win32 Synchronization Sample Application - hit any key to
exit...\n");

/* create the reset events */

for (Index = 0; Index < MAX_RESET_EVENTS; Index++) {
hEvent[Index] = CreateEvent((LPSECURITY_ATTRIBUTES)NULL,
FALSE,
FALSE,
NULL);
}

/* create one thread to process all the voice channels */

if ((thread_handle = (HANDLE)_beginthreadex(NULL,
0,
sample_begin,
(LPVOID)NULL,
0,
&threadID)) == (HANDLE)-1) {
printf("ERROR: Could not create thread : errno = %d\n", errno);
exit(1);
}

/* wait for Keyboard input to shutdown program */

getch();

Kbhit_flag++; /* let thread know it's time to abort */

SetEvent( hEvent[KEYBOARD_KEY] );

/* sleep here until thread has terminated */

if (WaitForMultipleObjects(1, &thread_handle, TRUE, INFINITE)
== WAIT_FAILED) {
printf("ERROR: Failed WaitForMultipleObjects(): error = %ld\n",
GetLastError());
}

return(0);
}

/*****
* NAME : DWORD WINAPI sample_begin(LPVOID argp)
* DESCRIPTION : do all channel specific processing
* INPUT : LPVOID argp - NULL pointer (not used)
*****/

```

```

* OUTPUT : none

* RETURNS : 0 on success; 1 if a failure was encountered

* CAUTIONS : none

*****/

DWORD WINAPI sample_begin(LPVOID argp)
{
    char channname[20];
    int chdesc;
    int cnt;
    int hDevice[MAX_CHAN];
    long EventCode;
    int Index;
    SRLWIN32INFO SrlWin32Info;

    /*
    * First thing is to inform SRL to signal the reset event
    * when a SCT event occurs
    */

    SrlWin32Info.dwTotalSize = sizeof(SRLWIN32INFO);
    SrlWin32Info.dwHandleType = SR_RESETEVENT;
    SrlWin32Info.ObjectHandle = hEvent[SCT_KEY];
    for (cnt = 0; cnt < MAX_CHAN; cnt++) {
        /* build name of voice channel */
        sprintf(channname, "vpmB%dC%d", (cnt / 4) + 1,
            (cnt % 4) + 1);
        /* open voice channel */
        if ((chdesc = vpm_open(channname, 0)) == -1) {
            printf("%s - FAILED: vpm_open(): errno = %d\n",
                channname, vpm_fileerrno());
            return(1);
        }
        hDevice[cnt] = chdesc;
        printf("%s - Voice channel opened\n", ATDV_NAMEP(chdesc));
        /* kick off the state machine by going offhook asynchronously */
        if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
            printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
                ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
            return(1);
        }
    }
}

```

```

printf("%s - Voice channel off-hook initialized\n",ATDV_NAMEP(chdesc));
}

/* loop until Keyboard input is received */
while (!Kbhit_flag) {
/*
* Wait on the reset events
*/
if ((Index = WaitForMultipleObjects(MAX_RESET_EVENTS,
hEvent,
FALSE,
INFINITE)) == WAIT_FAILED) {
printf("ERROR: Failed WaitForMultipleObjects(): error = %ld\n",
GetLastError());
}

/* check if it is because of a key hit */
if (Index == KEYBOARD_KEY) {
continue;
}

/* must be a SCT event so process it */
sr_waitevt(0);
/*
* gather data about the event
*/
chdesc = sr_getevtdev(0);
EventCode = sr_getevttype(0);
switch(EventCode) {
case TDX_SETHOOK:
if (VPMX_HOOKST(chdesc) == DX_OFFHOOK) {
printf("%s - Voice channel off-hook\n",ATDV_NAMEP(chdesc));
/* we went off hook so start dialing */
if (vpm_dial(chdesc, "12025551212", NULL, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_dial(): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
printf("%s - Voice channel dialing initialized\n",ATDV_NAMEP(chdesc));
} else {
/* we went on hook so go off hook again */

```

```

printf("%s - Voice channel on-hook\n",ATDV_NAMEP(chdesc));

/* set the voice channel off-hook */
if (vpm_sethook(chdesc, DX_OFFHOOK, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_sethook(DX_OFFHOOK): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}

printf("%s - Voice channel off-hook initialized\n",ATDV_NAMEP(chdesc));
}

break;

case TDX_DIAL:
printf("%s - Voice channel Done dialing\n",ATDV_NAMEP(chdesc));
/* done dialing so set the voice channel on-hook */
if (vpm_sethook(chdesc, DX_ONHOOK, EV_ASYNC) == -1) {
printf("%s - FAILED: vpm_sethook(DX_ONHOOK): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}

printf("%s - Voice channel on-hook initialized\n",ATDV_NAMEP(chdesc));
break;

default:
printf("Received unexpected event 0x%X on device %d\n", EventCode, chdesc);
break;
}
}

for (cnt = 0; cnt < MAX_CHAN; cnt++) {
/* close the voice channel */
chdesc = hDevice[cnt];
printf("%s - Voice channel closing\n",ATDV_NAMEP(chdesc));
if (vpm_close(chdesc) == -1) {
printf("%s - FAILED: vpm_close(): %s (error #%d)\n",
ATDV_NAMEP(chdesc), ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
return(1);
}
}

return(0);
}

```

SCT Standard Runtime Library

SRL event management device application management

Event data retrieval : event

Event management . Section 6.4. SRL Management Function

6.1.1. Event Handling Functions

```

device          event          , event handler
.
event          , backup event
handler        .

```

3 programming model Basic SRL

programming Models .

6.1.2. Event Data Retrieval Functions

sr_getevtdev()	·	event가 device handle
sr_getevttype()	·	event event type
sr_getevtlen()	·	event data length
sr_getevtdatap()	·	event pointer

Event data retrieval event . data extraction
event processing .

6.1.3. SRL Parameter Functions

sr_getboardcnt()	·	Get the number of boards of a specific type
sr_libinit()	·	Initialize the Standard Runtime Library DLL
sr_GetDllVersion()	·	Get the Standard Runtime Library DLL Version Number

6.2. SRL Error Handling

SRL event management .
· -1 .
· 1 .
NOTE : event 가
sr_getevtdatap() NULL .

가 error 7 SRL Standard Attribute Function
ATDV_LASTERR() ATDV_ERRMSGP() . SRL
가 argument SRL_DEVICE 가 ATDV_LASTERR()
error . Error text description ATDV_ERRMSGP .

error list .
ATDV_LASTERR() error가 ESR_SYS , windows 2000 system error
가 , errno.h global variable errno check .

Error code srl.lib.h Table 3 .

Table 3. Event Management Function Errors

Error Define	Error String
ESR_DATASZ	· Invalid size for default event data memory

Name: long sr_dishdlr(dev, evt_type, handler)	
Inputs: long dev	· SCT device handle
long evt_type	· event type
long(*handler)(unsigned long parm)	· event handling function
Returns: 0 if success	
-1 if failure	
Includes: srlib.h	
Type: event Handling function	

sr_dishdlr()	device/event type/handler triplet	sr_enbhdlr()
가	handler(), handler	· handler가
	sr_dishdlr() handler	·
		·

Parameter Description

dev	xx_open()	open device	handle	·
	xx open device identify	prefix	· device	
	device event		EV_ANYDEV	
	·			
evt_type	application	event	·	
	· event	· 가 event		
	technology-specific programmer ' s Guide			
	· evt_type EV_ANYEVT	· device	device	
	event	·		
	· evt_type EV_ANYEVT dev EV_ANYDEV	·		
	device	event	·	
handler	event	application-defined event handler	·	
	application event handler			
	sr_enbhdlr()	·		
handler		· handler		
		·		

Example

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

long int vpm_handler(unsigned long evhandle)
{
    printf( "vpm_handler() called, event is 0x%x\n", sr_getevtttype(evhandle));
    return( 0 ); /* tell SRL to dispose of the event */
}

main()
{
    int dxxxdev;
    int mode = SR_POLLMODE;
    /* open dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmBlC1", 0 )) == -1 ){
        printf( "vpm_open failed\n" );
        exit( 1 );
    }
    /* enable handler vpm_handler on device dxxxdev ..... */
    if( sr_enbhdlr( dxxxdev, EV_ANYEVT, vpm_handler ) == -1 ){
        printf( "Error: could not enable handler\n" );
        exit( 1 );
    }
    /* Disable the handler */
    if( sr_dishdlr( dxxxdev, EV_ANYEVT, vpm_handler ) == -1 ){
        printf( "Error: could not disable handler\n" );
        exit( 1 );
    }
}
```

Errors

error	1	,
ATDV_LASTERR(SRL_DEVICE)	.	ESR_SYS
가 error value	errno.h	errno

EINVAL · The dev/evt/handler triplet was not registered.

- `sr_enbhdr()`
- The appropriate technology-specific Programmer ' s Guide.

sr_enbhdr()

enables the handler function

Name: long sr_enbhdr(dev, evt_type, handler)

Inputs: long dev

· SCT device handle

long evt_type

· event type

long(*handler)(unsigned long parm)

· event handling function

Returns: 0 if success

-1 if failure

Includes: srllib.h

Type: Event Handling function

Description

sr_enbhdr() device/event handler handler 가 .

Handler device event SRL

parameter

.

Parameter Description

dev	open	,device	xx가 prefix	xx_open()
		device가 open		valid device
	handler	. device	event가	
	EV_ANYDEV	.		
evt_type	application	event	.	,
	· event	. 가	event type	
	technology-specific programmer ' s guide			.
	· device	event		dev parameter
	device evt_type	EV_ANYEVT	.	
	· device	event		evt_type
	EV_ANYEVT	dev parameter	EV_ANYDEV	.
Handler	event		application event handler	.

Event가 handler가 event SRL

handler . device event

general handler 가 , handler device

event 가 .

action handler .

- technology atomic
- technology multitasking
- device open close
- event handler
- handler , event handler .

Handler .

- technology multitasking
- sr_waitevt()

handler SRL event 0, 1 . Handler 1
 , SRL event , 0 , event .

event sr_waitevt() . SRL handler

2.5 SRL Extended Asynchronous Programming .

SRL event handling 가 handler
 handler . handler device open
 close . handler sr_waitevt()
 handler handler . Control sr_waitevt()가
 main thread .

event가 handler 0 sr_waitevt()

handler가 device event

handler가 handler event
 handler sr_waitevt()가 .

event 가 device가 handler close handler
 . event device service

handler가 event , handler가 release
 event handler event가 .

technology-specific programmer ' s Guide .

Example

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

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

main()
{
    int dxxxdev;
    int mode = SR_POLLMODE;

    /* open dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "vpm_open failed\n" );
        exit( 1 );
    }

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

Errors

가 error	1	
ATDV_LASTERR(SRL_DEVICE)	.	ESR_SYS
가	errno.h	errno
ENOMEM	.	event Library data structure
EINVAL	.	dev/evt/handler handler가

See Also

- sr_dishdlr()
- The appropriate technology-specific Programmer ' s Guide


```

#include <dxxilib.h>

long chdev[MAXDEVS];
long evt_handle;

main( ... )
{
    char channel_name[12], board_name[12];
    int brd_handle;

    int brd, ch, devcnt = 0;
    int numvoxbrds = 0;

    if ( sr_getboardcnt(vpm, &numvoxbrds) == -1) {
        /* error retrieving voice boards */
    }

    for (brd = 1; brd <= numvoxbrds; brd++) {
        /* build the board name and open the board device to get number of channels */
        sprintf(board_name, "vpmB%d\n", brd);
        if ( (brd_handle = vpm_open(board_name, 0)) == -1) {
            /* Board open error */
        }

        for (ch = 1; ch <= ATDV_SUBDEVS(brd_handle); ch++) {
            sprintf(channel_name, "%sC%d", board_name, ch);
            if ( (chdev[devcnt++] = vpm_open(channel_name, 0)) == -1) {
                /* Channel open error */
            }
        }

        /* End of channel for loop */
        vpm_close(brd_handle);
    } /* End of board loop */
}

```

sr_GetDllVersion() returns the SRL DLL Version Number

Name: long sr_GetDllVersion(dwfileverp, dwprodverp)

Inputs: LPDWORD dwfileverp · SRL DLL Version Number
 LPDWORD dwprodverp · Product version of this release

Returns: 0 if success
 -1 if failure

Includs: srl.lib.h

Description

sr_GetDllVersion()	file	product	SRL DLL Version Number	.
--------------------	------	---------	------------------------	---

	parameter	.
--	-----------	---

Parameter	Description
-----------	-------------

dwfileverp	file version information	pointer
dwprodverp	product version information	pointer

Example

```
/*$ sr_GetDllVersion( ) example $*/
#include <windows.h>
#include <srl.lib.h>
int InitDevices( )
{
    DWORD dwfilever, dwprodver;
    /*****
    * Initialize all the DLLs required. This will cause the DLLs to be
    * loaded and entry points to be resolved. Entry points not resolved
    * are set up to point to a default not implemented function in the
    * 'C' library. If the DLL is not found all functions are resolved
    * to not implemented.
    *****/
    if (sr_libinit(DLGC_MT) == -1) {
        /* Must be already loaded, only reason if sr_libinit( ) was already called */
    }
    /*****/
}
```



```

/* SRL library initialized so all other SRL functions may be called as normal.
* Display the version number of the DLL
*****/
sr_GetDllVersion(&dwfilever, &dwprodver);
printf("File Version for SRL is %d.%02d\n",
HIWORD(dwfilever), LOWORD(dwfilever));
printf("Product Version for SRL is %d.%02d\n",
HIWORD(dwprodver), LOWORD(dwprodver));
/* Call technology specific xx_libinit( ) functions */
}

```

Caution

Cross-Compatibility Library

.

See Also

sr_getevtdatap() returns the address of the variable data block

Name:	void *sr_getevtdatap(ehndadle)		
Inputs:	unsigned long	· sr_waitevtEx()	event handle
	Ehandle	, sr_waitevt()가	0
Returns:	data가	, data block	, data가 NULL
Includes: srllib.h			
Type:	Event Data Retrieval function		

Description			
sr_getevtdatap()	event	data block	address
Event data	data pointer	event length sr_getevtlen()	.
가	가	data	event NULL
.			

NOTE: data the appropriate technology-specific programmer ' s Guide .

Cautions
sr_waitevt() application sr_getevtdatap() parameter 0 .

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxplib.h>
long chdev[MAXDEVS];
unsigned long evt_handle;
main( ... )
{
    char channel_name[12];
    int ch;
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Build the channel name for each channel */
        if ( (chdev[ch] = vpm_open(channel_name, 0) ) == -1 ) {
            printf("vpm_open failed\n");
            exit(1);
        }
    }
}
```

```

    }

    /*
    * Now initialize each device setting up the event masks and then issue
    * the command asynchronously to start off the state machine.
    */

    for (ch = 0; ch < MAXDEVS; ch++) {

        /* Set up the event masks and other initialization */

        /* set the channel onhook asynchronously */
        if (vpm_sethook( chdev[ch]. DX_ONHOOK, EV_ASYNC) == -1) {

            /* sethook failed, handle the error */

        }

    }

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

        /* wait for the event */

        sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);

        process_event( evt_handle);

    }

int process_event( ehandle)
unsigned long ehandle;
{

    int voxhandle = sr_getevtdev(ehandle);
    int *datap;
    switch(sr_getevtttype(ehandle)) {
    case TDX_CST:

        *datap = (int *) sr_getevtdatap(ehandle);
        if (datap->csd_data == DE_RINGS)
        {
            .
            .
        }
        break;
    case TDX_PLAY:
        .
        break;
    }
}
}

```

See Also

- `sr_getevtlen()`
- The appropriate technology-specific Programmer ' s Guide

sr_getevtdev() returns the SCT device handle

Name: long sr_getevtdev(ehandle)

Inputs: unsigned long ehandle · sr_waitevtEx() event handle
, sr_waitevt()가 0

Returns: SCT device handle, event가 1

Includes: srllib.h

Type: Event Data Retrieval function

Description

sr_getevtdev() event SCT device handle · timeout
event , SRL_DEVICE ·

NOTE: device handler the
appropriate technology-specific programmer ' s Guide .

Cautions

sr_waitevt() application sr_getevtdatap() parameter 0

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxplib.h>
long chdev[MAXDEVS];
long evt_handle;
main( ... )
{
    char channel_name[12];
    int ch;
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Build the channel name for each channel */
        if ( (chdev[ch] = vpm_open(channel_name, 0) ) == -1 ) {
            printf("vpm_open failed\n");
            exit(1);
        }
    }
    /*
```

```

    * Now initialize each device setting up the event masks and then issue
    * the command asynchronously to start off the state machine.
    */
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Set up the event masks and other initialization */
        /* set the channel onhook asynchronously */
        if (vpm_sethook( chdev[ch]. DX_ONHOOK, EV_ASYNC) == -1) {
            /* sethook failed, handle the error */
        }
    }

    /* This is the main loop to control the Voice hardware */
    while (FOREVER) {
        /* wait for the event */
        sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);
        process_event( evt_handle);
    }

int process_event( ehandle)
unsigned long ehandle;
{
    int voxhandle = sr_getevtddev(ehandle);
    switch(sr_getevtttype(ehandle)) {
    case TDX_CST:
        .
        break;
    case TDX_PLAY:
        .
        break;
    }
}

```

See Also

- [sr_waitevt\(\)](#)
- [sr_waitevtEx\(\)](#)
- [The appropriate technology-specific Programmer ' s Guide](#)


```

        printf("vpm_open failed\n");
        exit(1);
    }
}

/*
 * Now initialize each device setting up the event masks and then issue
 * the command asynchronously to start off the state machine.
 */
for (ch = 0; ch < MAXDEVS; ch++) {
    /* Set up the event masks and other initialization */
    /* set the channel onhook asynchronously */
    if (vpm_sethook( chdev[ch]. DX_ONHOOK, EV_ASYNC) == -1) {
        /* sethook failed, handle the error */
    }
}

/* This is the main loop to control the Voice hardware */
while (FOREVER) {
    /* wait for the event */
    sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);
    process_event( evt_handle);
}

int process_event( ehandle)
unsigned long ehandle;
{
    long varlen
    int voxhandle = sr_getevtdev(ehandle);
    switch(sr_getevtttype(ehandle)) {
    case TDX_CST:
        varlen = sr_getevtlen(ehandle)
        break;
    case TDX_PLAY:
        .
        break;
    }
}
}

```

See Also

- The appropriate technology-specific Programmer ' s Guide

sr_getevtttype() return the event type for the current event

Name: long sr_getevtttype(ehandle)

Inputs: unsigned long · sr_waitevtEx() event handle
 Ehandle , sr_waitevt()가 0

Returns: event type, event가 1

Includes: srllib.h

Type: Event Data Retrieval function

Description

sr_getevtttype() event event type . event
 timeout SR_TIMEOUTEVT .

NOTE: the device-specific event type the
 appropriate technology-specific programmer ' s Guide .

Cautions

sr_waitevt() sr_waitevtEx() .

sr_waitevt() application parameter 0 , sr_getevttlen(0)
 . Ehandle event descriptor handle .

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxplib.h>
long chdev[MAXDEVS];
unsigned long evt_handle;
main( ... )
{
    char channel_name[12];
    int ch;
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Build the channel name for each channel */
        if ( (chdev[ch] = vpm_open(channel_name, 0) ) == -1 ) {
            printf("vpm_open failed\n");
            exit(1);
        }
    }
}
```

```

        }

    }

    /*
    * Now initialize each device setting up the event masks and then issue
    * the command asynchronously to start off the state machine.
    */

    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Set up the event masks and other initialization */
        /* set the channel onhook asynchronously */
        if (vpm_sethook( chdev[ch]. DX_ONHOOK, EV_ASYNC) == -1) {
            /* sethook failed, handle the error */
        }
    }

    /* This is the main loop to control the Voice hardware */
    while (FOREVER) {
        /* wait for the event */
        sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);
        process_event( evt_handle);
    }

int process_event( ehandle)
unsigned long ehandle;
{
    int voxhandle = sr_getevtdev(ehandle);
    switch(sr_getevttype(ehandle)) {
    case TDX_CST:
        .
        break;
    case TDX_PLAY:
        .
        break;
    }
}

```

See Also

- `sr_waitevt()`
- `sr_waitevtEx()`
- The appropriate technology-specific Programmer ' s Guide

```
send event notification to a window
```

Inputs:	HWND handle	·	Window handle
	Unsigned int message	·	window message number
	Unsigned int flags	·	notification on/off flag
Returns:	0, 1		
Includes:	srllib.h		
Type:	Event handling		

```

sr_NotifyEvent()      SRL      windows      event      .      window event
queue      SCT event      ,      actual event가 window ' s
event queue      event가      , SCT event
queue      event      sr_waitevt()      application      notification
message가      .

```

handle	Window handle to which the message is to be sent	
message	message number (message ID)	
flag	flags to turn event notification on or off:	
	SR_NOTIFY_ON	event notification on
	SR_NOTIFY_OFF	event notification off

```

thread          device  control          application
.  application  SCT evetn queue      event
sr_waitevt(0)  .

```

2. Event가 window message 가 window message 가 .

```
#include <windows.h>

#include <srllib.h>
```

```

#include <dxxilib.h>

#define WM_SRNOTIFYEVENT WM_USER + 100 /* user defined message for event
notification */

long PASCAL FrameWndProc(HWND, UINT, UINT, LONG);

WNDCLASS wndclass;

char szFrameClass[] = "MdiFrame";

HWND hwndFrame;

main( ... )
{
    int chdev;

    .

    .

    .

    /* Register window class */
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = FrameWndProc;

    .

    .

    .

    wndclass.lpszClassName = szFrameClass;
    RegisterClass(&wndclass);
    /* Create Frame window */
    hwndFrame = CreateWindow(szFrameClass,
        "Sample SCT Voice Application",
        WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL);

    /* Turn on event notification as Window message */
    sr_NotifyEvent(hwndFrame,WM_SRNOTIFYEVENT, SR_NOTIFY_ON);

    .

    .

    .

```

```

        if ((chdev = vpm_open("dxxB1C1",0)) == -1) {
            MyPrintf("Failed to open dxxB1C1\n");
            exit(0);
        }

        if (vpm_sethook(chdev,DX_ONHOOK,EV_ASYNC) == -1) {
            MyPrintf("Failed to go offhook: %s\n",ATDV_ERRMSGP(chdev));
        }
        .
        .
        .
    }

/*****
* NAME: FrameWndProc()
* DESCRIPTION: Frame window procedure in an MDI application.
*****/
long PASCAL FrameWndProc(HWND hwnd, UINT message,
UINT wParam, LONG lParam)
{
    switch(message) {
        case WM_CREATE:
            .
            .
            .

        case WM_COMMAND:
            .
            .
            .

        case WM_SRNOTIFYEVENT:
            if (sr_waitevt(0) == -1) {
                MyPrintf("sr_waitevt: %s",ATDV_ERRMSGP(SRL_DEVICE));
                break;
            }
            switch( sr_getevtttype()) {
                case TDX_SETHOOK:
                    MyPrintf("Sethook complete\n");
                    break;

                case TDX_PLAY:

                case TDX_RECORD:

```

```

        .
        .
        .
    }
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    return 0;
}
.
.
.
}

```

See Also

- [sr_waitevt\(\)](#)
- [sr_getevtdev\(\)](#)
- [sr_getevttype\(\)](#)
- [sr_getevtlen\(\)](#)
- [sr_getevtdatap\(\)](#)
- The appropriate technology-specific Programmer ' s Guide

Name : long sr_putevt(dev , evttype , evtlen , evtdatap , errcode)

- Input :
- long dev · device
 - Unsigned long evttype · event type ID
 - Long evtlen · data block pointer
 - Void *evtdatap · data block pointer
 - Long errcode · error code

Return : 0
 1

Include : srllib.h

 Dxxxlib.h

Type : Event Management function

Mode : Synchronous

Description				
sr_putevt()	Application	SRL event queue	event	
.	event data가	evtdatap parameter	evtlen parameter	
	.	SRL evtdatap parameter		
data	.	caller sr_putevt()	calling	event
data	.			
	parameter	.		

Parameter	Description			
dev	SRL_DEVICE	dev parameter	xx_open()	
	calling	open	device handle	.sr_getevtdev()
	event가	.		
Evttype	가	event type.	event	.
	Sr_getevttype()	event가	.	
Evtlen	event	data	length.event가	
		sr_getevtlen()	.	event
	data가	evtlen 0	evtdatap parameter	NULL
	.			
Evtdatap	evtlen parameter	size	pointer.	SRL
	Data	event	.sr_getevtdatap()	
		memory	pointer	return .

event data가 parameter NULL
 evtlen parameter 0 .
 Errcode event error code . error return
 event device handle ATDV_LASTERR() calling .
 0 device error가
 . non-zero device ATDV_LASTERR()
 error .

sr_waitevt() sr_waitevt(Ex) sr_putevt() event
 . sr_putevt() event block
 unblock synchronous .
 Voice library vpm_getevt() vpm_wtring() 가 queue CST
 event queue TDX_CST event incoming call .

Cautions

device handle .

Example

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

int dev;          /*SCT device handle*/
DX_CST cst;      /*TDX_CST event data block*/

/* Open board 1 channel 1 device*/
if ((dev = vpm_open("vpmB1C1" , 0)) == -1){
    printf("Cannot open channel vpmB1C1");
}

/*Simulate an incoming call*/
cst.cst_event = DE_RINGS;
cst.cst_data = 0;

/*Put the event on the event queue*/
if (sr_putevt(dev,TDX_CST, sizeof(DX_CST) , &cst,0) != SR_SUCCESS)
{
    printf("_sr_putevt_ dailed - %s", ATDV_ERRMSGP(SRF_DEVICE));
}
```

```
.  
.   
.   
}
```

Errors

sr_putevt() memory가 SRL event queue
SR_NOMEM .

See Also

•sr_waitevt()

wait for any event to occur

sr_waitevt()

Name: long sr_waitevt(timeout)

Inputs: long timeout . timeout(msec)

Returns: timeout , timeout 1

Includes: srllib.h

Type: Event control function

Description

sr_waitevt() SCT device event

Parameter Description

timeout -1 timeout sr_waitevt가 event .
SRL event queue event가
SRL event가 timeout 0 .
sr_waitevt() event가 timeout .
timeout event 1 . 가
timeout timeout event SRL_DEVICE . sr_getevtdev(0)
event SCT device handle SRL_DEVICE . sr_getevttype(0)
SR_TMOUTEVT SR_TMOVEVT event type .

Cautions

application event event , event
sr_waitevt() . sr_waitevt()
event event .

sr_waitevt() handler .

thread sr_waitevt() sr_waitevtEx() .

event sr_waitevt()

parameter 0 pass .

- sr_getevtdatap(0)
- sr_getevtdev(0)

- sr_getevtlen(0)
- sr_getevttype(0)

Example

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

int vpm_handler(unsigned long evhandle)
{
    printf( "Got event 0x%x on device %s, data length = %d, datap = 0x%x\n",
        sr_getevttype(evhandle), ATDV_NAMEP( sr_getevtdev()), sr_getevtlen(evhandle),
        sr_getevtdatap(evhandle));

    /* Tell SRL to keep the event */
    return( 1 );
}

main()
{
    int dxxxdev;
    int mode = SR_STASYNC;

    /* Open a dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }

    /* enable a handler for all events on the device */
    if( sr_enbhdlr( dxxxdev, EV_ANYEVT, vpm_handler ) == -1 ){
        printf( "Error: could not enable handler\n" );
        exit( 1 );
    }

    /* Perform an async function on the device */
    if( vpm_sethook( dxxxdev, DL_ONHOOK, EV_ASYNC ) == -1 ){
        printf( "vpm_sethook failed: error = %s\n", ATDV_ERRMSGP( dxxxdev ));
        exit( 1 );
    }

    /* Wait 10 seconds for an event */
    if( sr_waitevt( 10000 ) == -1 ){
        printf( "sr_waitevt, %s\n",
```

```

        ATDV_ERRMSGP( SRL_DEVICE ));
        exit( 1 );
    }
    /* Disable the handler */
    if( sr_dishdlr( dxxxdev, EV_ANYEVT, vpm_handler ) == -1 ){
        printf( "Error: could not disable handler\n" );
        exit( 1 );
    }
    if( vpm_close( dxxxdev ) == -1 ){
        printf( "Error: could not close dxxxdev\n" );
        exit( 1 );
    }
    exit( 0 );
}

```

Errors

가 error 1 ,
 ATDV_LASTERR(SRL_DEVICE) . ESR_SYS
 가 errno errno
 EINVAL · Invalid timeout value

sr_waitvtEx() waits for events on certain devices

Name: long sr_waitvtEx(handlep, handlecnt, timeout, event_handler)

Inputs: long *handlep · event가 handle array
pointer

int handlecnt · handle handle

long timeout · millisecond timeout value

long *event_handler · event handle pointer

Returns: 0, timeout 1

Includes: srllib.h

Type: Event Control function

Description

sr_waitvtEx() device event · standard
sr_waitvt(), · handle event
가 · xx_open device
handle event ·

Parameter Description

handlep open handle array 가 ·
handlecnt device가 array ·
timeout -1 timeout event sr_waitvt()가
· Event가 SRL event queue
SRL event가 timeout 0 ·
event_handler event가 event handle event_handler argument
application · event sr_getvtdev(),
sr_getvttype(), sr_getvtlen() sr_getvtdata()

sr_waitvtEx() single-threaded multithreaded application
· , thread sr_waitvtEx() ,
thread sr_waitvtEx() event 가 · Application
device event sr_waitvtEx() block multiple thread 가
· thread가 event pickup ·
thread state machine ·

Cautions

sr_waitevt()	sr_waitevtEx()	application	.
sr_waitevtEx()	handler	.	.

Example

```
#include <srllib.h>
#include <windows.h>
#include <dxxxlib.h>
long chdev[MAXDEVS];
unsigned long evt_handle;
main( ... )
{
    char channel_name[12];
    int ch;
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Build the channel name for each channel */
        if ( (chdev[ch] = vpm_open(channel_name, 0) ) == -1 ) {
            printf("vpm_open failed\n");
            exit(1);
        }
    }
    /*
    * Now initialize each device setting up the event masks and then issue
    * the command asynchronously to start off the state machine.
    */
    for (ch = 0; ch < MAXDEVS; ch++) {
        /* Set up the event masks and other initialization */
        /* set the channel onhook asynchronously */
        if (vpm_sethook( chdev[ch]. DX_ONHOOK, EV_ASYNC) == -1) {
            /* sethook failed, handle the error */
        }
    }
    /* This is the main loop to control the Voice hardware */
    while (FOREVER) {
        /* wait for the event */
        sr_waitevtEx( chdev, MAXDEVS, -1, &evt_handle);
        process_event( evt_handle);
    }
}
```

```

    }

int process_event( ehandle)
unsigned long ehandle;
{
    int voxhandle = sr_getevtdev(ehandle);
    switch(sr_getevttype(ehandle)) {
        case TDX_CST:
            .
            break;
        case TDX_PLAY:
            .
            break;
    }
}

```

Errors

error 가 1 ,
 ATDV_LASTERR(SRL_DEVICE) . ESR_SYS
 가 errno.h errno .
 ESR_TMOUT · Timed out waiting for event

See Also

- sr_waitevt()
- sr_getevtdev()
- sr_getevttype()
- sr_getevtlen()
- sr_getevtdatap()
- The appropriate technology-specific Programmer ' s Guide

Library	Windows 2000 Standard reference	Attribute function	SCT standard Runtime

standard attribute function	windows 2000(SRL)	SCT standard Runtime
.	SCT device	device name library
error	.	

- function name
- function name “ ATDV__ ” 가
- name attribute

ATDV_ERRMSGP()	· pointer to string describing error on last library call
ATDV_LASTERR()	· library call device error
ATDV_NAMEP()	· device name pointer
ATDV_SUBDEVS()	· subdevice

code	line	standard attribute function	application
		.	

DEVICElib.h	device	header file	.
NOTE: srllib.h	SCT header file	library	code
	device가 D/4x	dxxplib.h file	

function	device
----------	--------

Specific device
specific programmer ' s Guide

the appropriate technology-

.

ATDV_ERRMSGP() returns a pointer to an ASCIIZ string

Name:	char * ATDV_ERRMSGP(dev)
Inputs:	int dev . valid SCT device handle
Returns:	pointer to string
Includes:	srllib.h
Category:	standard attribute

Description			
ATDV_ERRMSGP()	device	error	ASCIIZ string
pointer	. pointer	application	.
function	device	error	, string
No Error	.		

Parameter	Description
-----------	-------------

dev	Device handle, the handle returned by the technology-specific xx_open()
-----	--

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxplib.h>
main()
{
    int dxxdev;
    int parm = ET_RON;
    /* Open dxxx channel device */
    if(( dxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
    /*Attempt to set a board level parameter on a channel device-will fail */
    if( vpm_setparm( dxxdev, DXBD_R_EDGE, &parm ) == -1 ){
        printf( "The last error on the device was '%s'\n",
            ATDV_ERRMSGP( dxxdev ));
    }
```

}

Errors

```
dev    invalid device handle      ,      “ unknown device ”      string
      pointer                      .
```

See Also

- The appropriate technology-specific Programmer ' s Guide

ATDV_LASTERR() indicates the last error that occurred

Name: long ATDV_LASTERR(dev)

Inputs: int dev . valid SCT device handle

Returns: invalid device handle , AT_FAILURE
valid error number

Includes: srlLib.h

Category: standard attribute

Description

ATDV_LASTERR()	device	error	long
. Error	device	technology specific header file	.

Parameter	Description
-----------	-------------

dev	Device handle, the handle returned by the technology-specific xx_open()
-----	--

Example

```
#include <windows.h>
#include <srlLib.h>
#include <dxxxLib.h>
main()
{
    int dxxxdev;
    /* Open a dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
    printf( "Device irq is %d\n", ATDV_IRQNUM( dxxxdev ));
}
```

Errors

device handle dev , function AT_FAILURE

.

See Also

- The appropriate technology-specific Programmer ' s Guide

ATDV_LASTERR() indicates the last error that occurred

Name: long ATDV_LASTERR(dev)
 Inputs: int dev · valid SCT device handle
 Returns: invalid device handle , AT_FAILURE
 valid error number
 Includes: srlLib.h
 Category: standard attribute

Description

ATDV_LASTERR()	device	error	long
. Error	device	technology specific header file	.

Parameter	Description
-----------	-------------

dev	Device handle, the handle returned by the technology-specific xx_open()
-----	--

Example

```
#include <windows.h>
#include <srlLib.h>
#include <dxxxLib.h>
main()
{
    int dxxxdev;
    int parm = ET_RON;
    /* Open dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
    /*Attempt to set a board level parameter on a channel device-will fail */
    if( vpm_setparm( dxxxdev, DXBD_R_EDGE, &parm ) == -1 ){
        printf( "The last error on the device was 0x%x\n",
            ATDV_LASTERR( dxxxdev ));
    }
}
```

Errors

`device handle dev`, function `AT_FAILURE`.

See Also

- The appropriate technology-specific Programmer ' s Guide

ATDV_NAMEP() return a pointer to an ASCIIZ string

Name:	char * ATDV_NAMEP(dev)
Inputs:	int dev · valid SCT device handle
Return:	pointer to string
Includes:	srllib.h
Category:	standard attribute

Description			
device	open	device name	ASCIIZ string
ATDV_NAMEP()	pointer	.	
· vpmBbCc			
where			
· b : system board			
· c : VPM board channel			

device가 open string remain pointer가 .

Parameter	Description
-----------	-------------

dev	Device handle, the handle returned by the technology-specific xx_open()
-----	--

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxxlib.h>
main()
{
    int dxxxdev;
    /* Open a dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1C1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
    printf( "Device name is %s\n", ATDV_NAMEP( dxxxdev ));
}
```

Errors

device handle dev function " Unknown
device " string pointer .

See Also

- The appropriate technology-specific Programmer ' s Guide

`ATDV_SUBDEVS()` returns the number of subdevices for the device

Name:	long ATDV_SUBDEVS(dev)
Inputs:	int dev . valid SCT device handle
Returns:	, AT_FAILURE subdevice
Includes:	srllib.h
Category:	standard attribute

Description				
ATDV_SUBDEVS()	device	subdevice	.	integer
.				
Subdevice	VPM	SPM board	channel	.

Parameter	Description
dev	Device handle, the handle returned by the technology-specific xx_open()

Example

```
#include <windows.h>
#include <srllib.h>
#include <dxxplib.h>
main()
{
    int dxxxdev;
    /* Open a dxxx channel device */
    if(( dxxxdev = vpm_open( "vpmB1", 0 )) == -1 ){
        printf( "Error: cannot open device\n" );
        exit( 1 );
    }
    printf( "Device has %d subdevices\n", ATDV_SUBDEVS( dxxxdev ) );
}
```

Errors

device handle dev , AT_FAILURE
.

See Also

- The appropriate technology-specific Programmer ' s Guide

8. DV_TPT Termination Parameter Table Structure

DV_TPT structures

Termination parameter table SCT device . srllib.h
가 DV_TPT structure

- DV_TPT structure linked list
- DV_TPT structure array
- DV_TPT structure linked list array

structure multitasking function

Description of the typedef for the DV_TPT

DV_TPT structure typedef

```
typedef struct DV_TPT {  
    unsigned short tp_type; /* Flags describing this structure */  
    unsigned short tp_termno; /* Termination parameter number */  
    unsigned short tp_length; /* Length of terminator */  
    unsigned short tp_flags; /* Term. parameter attributes flag */  
    unsigned short tp_data; /* Optional additional data */  
    unsigned short rfu; /* Reserved for future use */  
    struct DV_TPT *tp_nextp; /* Pointer to next term. parameter if */  
    /* IO_LINK is set */  
} DV_TPT;
```

where :

Parameter	Description
· tp_type	structure가 (IO_LINK), (IO_CONT), DV_TPT table(IO_EOT) DV_TPT entry
· tp_termno	.
· tp_length	size
· tp_flags	termno

· tp_data optional 가 data

· tp_nextp list DV_TPT structure pointer .

, VPM, SPM device field 가

. windows 2000 the voice Programmer ' s Guide

.

Glossary

Asynchronous function: application event
 function. EV_ASYNC function mode argument .
 function thread code .

Backup handlers: device event device
 event handler

Device: physical library object, board
 channel

Device handle: open device Numerical reference.
 Device name: xx개 device type open xx_open()
 function device access device
 reference

SCT Standard Runtime Library for Windows 2000 (SRL): event management
 function standard attribute function Device-independent library

Event: device message

Handler: event개 event , SRL user-
 defined function

Event management functions: device 개 event
 application-specified event handler event
 SRL function

Solicited event: library ' s asynchronous function .
 vpm_play() solicited event “ play ” .

Standard attribute functions: function call device
 SRL function. Standard attribute information SRL
 SCT device .

Subdevices: device direct child device board device
 subdevice . Subdevice개 device subdevice
 subdevice direct child .

Synchronous function: function application block function.
 EV_SYNC function ' s mode argument .

Unsolicited event: channel silence_on silence-off prompting
 event