USB20HR Windows API

User Guide

System Level Solutions, Inc. (USA)
14100 Murphy Avenue
San Martin, CA 95046
(408) 852 - 0067

http://www.slscorp.com

Windows API Version:	1.3
Document Version:	1.5
Document Date:	April 2008

IP Usage Note

The Intellectual Property (IP) core is intended solely for our clients for physical integration into their own technical products after careful examination by experienced technical personnel for its suitability for the intended purpose.

The IP was not developed for or intended for use in any specific customer application. The firmware/software of the device may have to be adapted to the specific intended modalities of use or even replaced by other firmware/software in order to ensure flawless function in the respective areas of application.

Performance data may depend on the operating environment, the area of application, the configuration, and method of control, as well as on other conditions of use; these may deviate from the technical specifications, the Design Guide specifications, or other product documentation. The actual performance characteristics can be determined only by measurements subsequent to integration.

The reference designs were tested in a reference environment for compliance with the legal requirements applicable to the reference environment.

No representation is made regarding the compliance with legal, regulatory, or other requirements in other environments. No representation can be made and no warranty can be assumed regarding the suitability of the device for a specific purpose as defined by our customers.

SLS reserves the right to make changes to the hardware or firmware or software or to the specifications without prior notice or to replace the IP with a successor model to improve performance or design of the IP. Of course, any changes to the hardware or firmware or software of any IP for which we have entered into an agreement with our customers will be made only if, and only to the extent that, such changes can reasonably be expected to be acceptable to our customers.

ug_ipusb20hr_winapi_1.5

About this Guide

Introduction

This guide helps users to develop software application on the top of USB20HR device driver.

Table below shows the revision history of the user guide.

Version	Date	Description
1.5	April 2008	• Changed layout design. • Replaced USB 2.0 with USB20HR
1.4	September 2007	• Added code for the board, Modified version history & Document part no.
1.3	March 2007	Make changes in Chapter 2 and modified as per USB2.0 Ver1.2
1.2	April 2006	3rd publication of the API guide- Generalize the API for USB1.1 and USB 2.0. Also change the naming convention of product code.
1.1	October 2005	Change layout design.
1.0.	May 2005	First Publication of the Application Guide.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Use Ctrl + F to open the Find dialog box. Use Shift + Ctrl + N to open to the Go To Page dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature preview of each page, provide a link to the pages.
- Numerous links shown in Navy Blue color allow you to jump to related information.

How to Contact SLS

For the most up-to-date information about SLS products, go to the SLS worldwide website at http://www.slscorp.com. For additional information about SLS products, consult the source shown below.

Information Type	Email
Product literature services, SLS literature services, Non-technical customer services, Technical support.	support@slscorp.com

Typographic Conventions

The user guide uses the typographic conventions as shown below:

Visual Cue	Meaning
Bold Type with Initial Capital letters	All headings and Sub headings Titles in a document are displayed in bold type with initial capital letters; Example: **SLS_ListDevices, Win32 API Reference.**
Bold Type with Italic Letters	All Definitions, Figure and Table/Example Headings are displayed in Italics. Examples: ***Figure 1-1. SLSUSB Driver Architecture***
Courier Type with Upper Case	All System variables or constatns are defined in Courier type with Upper Case; Example:**SLS_STATUS, PVOID**
Courier Type with Italics	All variables and parameters are described in Courier type with italics;Example: ***PvArg1,PvArg2***
Courier Type with Bold	All function definations are describes in Courier type with bold:Example: **SLS_W32CreateFile**
1., 2.	Numbered steps are used in a list of items, when the sequence of items is important. such as steps listed in procedure.
•	Bullets are used in a list of items when the sequence of items is not important.
☞	The hand points to special information that requires special attention
CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.

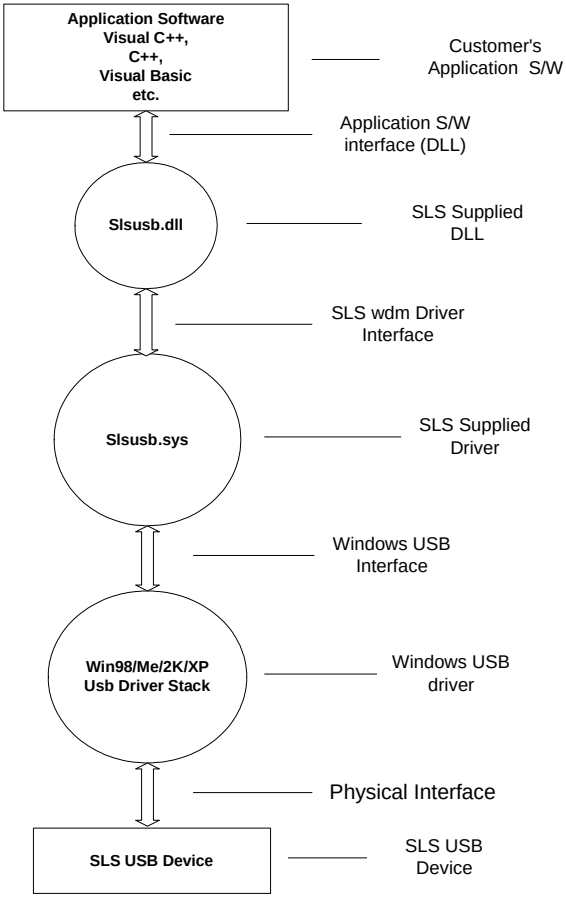
Visual Cue	Meaning
	The warning indicates information that should be read prior to starting or continuing the procedure or processes.
	The feet direct you to more information on a particular topic.

Contents

1. Introduction

The architecture of the USB20HR, the USB 2.0 device driver, consists of a Windows WDM driver that communicates with the device via the Windows USB Stack and a DLL which interfaces the Application Software (written in VC++, C++ Builder, Delphi, VB etc.) to the WDM driver.

Figure 1-1. ***USB20HR Device Driver Architecure***



2. API Reference

This section explains USB20HR Windows API functions.

SLS_ListDevices()

Protocol: `SLS_STATUS SLS_ListDevices (PVOID pvArg1, PVOID pvArg2, DWORD dwFlags)`

Description: Gets information concerning the devices currently connected. Returns information such as the number of devices connected and device strings such as serial number and product description.

Parameters:

- *pvArg1*-Meaning depends on *dwFlags*
- *pvArg2*-Meaning depends on *dwFlags*
- *dwFlags*-Determines format of returned information.Table below shows dwFlags and its description

Flag Name	Flag Description
SLS_LIST_NUMBER_ ONLY	Number of devices currently connected.
SLS_OPEN_BY_ SERIAL_NUMBER	Serial number string will be returned from the function.
SLS_OPEN_BY_ DESCRIPTION	the product description string will be returned from this function.
SLS_LIST_BY_ INDEX	return device string information for a single device.
SLS_LIST_ALL	device string information for all connected devices.

Return Value: `SLS_OK` if successful, otherwise the return value is an SLS error code.

Remarks: This function can be used in a number of ways to return different types of information.

- In its simplest form, it can be used to return the number of devices currently connected. If `SLS_LIST_NUMBER_ONLY` bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to a `DWORD` location to store the number of devices currently connected.
- It can be used to return device string information. If `SLS_OPEN_BY_SERIAL_NUMBER` bit is set in *dwFlags*, the serial number string will be returned from this function. If `SLS_OPEN_BY_DESCRIPTION` bit is set in *dwFlags*, the product description string will be returned from this function. If neither of these bits is set, the serial number string will be returned by default.

- It can be used to return device string information for a single device. If SLS_LIST_BY_INDEX bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a buffer to contain the appropriate string. Indexes are zero-based, and the error code SLS_DEVICE_NOT_FOUND is returned for an invalid index.
- It can be used to return device string information for all connected devices. If SLS_LIST_ALL bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings, and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected. Note that, for *pvArg1*, the last entry in the array of pointers to buffers should be a NULL pointer so the array will contain one more location than the number of devices connected.

Example 1: Example2-1. below shows how to get the number of devices currently connected.

Example 2-1. Getting the No. Of Devices Connected

```
SLS_STATUS SLSStatus;
DWORD numDevs;

SLSStatus = SLS_ListDevices(&numDevs,
                            NULL,SLS_LIST_NUMBER_ONLY);

if (SLSStatus == SLS_OK)
{
// SLS_ListDevices OK, number of devices connected is in numDevs
}
else
{
// SLS_ListDevices failed
}
```

Example 2: The Example2-2. shows how to get the serial number of the first device found. Note that indexes are zero-based. If more than one device is connected, incrementing devIndex will get the serial number of each connected device in turn.

Example 2-2. Getting Serial Number of the First Device Found

```
SLS_STATUS SLSStatus;
DWORD devIndex = 0;
char Buffer[16];

SLSStatus = SLS_ListDevices((PVOID)devIndex,Buffer,

SLS_LIST_BY_INDEX|SLS_OPEN_BY_SERIAL_NUMBER);

if (SLS_SUCCESS(SLSStatus))
{
// SLS_ListDevices OK, serial number is in Buffer
}
else
{
 // SLS_ListDevices failed
}
```

Example 3: The Example2-3. shows how to get the product descriptions of all the devices currently connected.

Example 2-3. Getting Product Descriptions of Currently Connected Devices

```
SLS_STATUS SLSStatus;
char *BufPtrs[3]; // pointer to array of 3 pointers
char Buffer1[64]; // buffer for product description of first device
found
char Buffer2[64]; // buffer for product description of second device
DWORD numDevs;

// initialize the array of pointers
BufPtrs[0] = Buffer1;
BufPtrs[1] = Buffer2;
BufPtrs[2] = NULL; // last entry should be NULL

SLSStatus = SLS_ListDevices(BufPtrs,&numDevs,
                      SLS_LIST_ALL|SLS_OPEN_BY_DESCRIPTION);


if (SLS_SUCCESS(SLSStatus))
{
 // SLS_ListDevices OK, product descriptions are in Buffer1 and
    Buffer2, and
 // numDevs contains the number of devices connected
}
else
{
 // SLS_ListDevices failed
}
```

SLS_W32_Create File()

Protocol:

SLS_HANDLE SLS_W32_CreateFile(LPCSTR *lpszName*, DWORD *dwAccess*, DWORD *dwShareMode*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*, DWORD *dwCreate*, DWORD *dwAttrsAndFlags*, HANDLE *hTemplate*)

Description:

Opens the named device and return a handle that will be used for subsequent accesses. The device name can be its serial number or device description.

Parameters:

lpszName. Pointer to a null terminated string that contains the name of the device. The name of the device can be its serial number or description as obtained from the SLS_ListDevices function.

dwAccess. Type of access to the device. Access can be GENERIC_READ, GENERIC_WRITE, or both.

dwShareMode. How the device is shared. This value must be set to 0.

lpSecurityAttributes. This parameter has no effect and should be set to NULL.

dwCreate. This parameter must be set to OPEN_EXISTING.

dwAttrsAndFlags. File attributes and flags. This parameter is a combination of FILE_ATTRIBUTE_NORMAL, FILE_FLAG_OVERLAPPED if overlapped I/O is used, SLS_OPEN_BY_SERIAL_NUMBER if *lpszName* is the device's serial number, and SLS_OPEN_BY_DESCRIPTION if *lpszName* is the device's description.

hTemplate. This parameter must be NULL.

Return Value:

If the function is successful, the return value is a handle. If the function is unsuccessful, the return value is the Win32 error code INVALID_HANDLE_VALUE.

Remarks:

This function must be used if overlapped I/O is required.

Example 1:

The Example2-4. example shows how a device can be opened for overlapped I/O using its serial number.

Example 2-4. Opening device for overlapped I/O using its serial number

```
SLS_STATUS SLSStatus;
SLS_HANDLE SLSHandle;
char Buf[64];

SLSStatus = SLS_ListDevices(0,Buf,SLS_LIST_BY_INDEX|
            SLS_OPEN_BY_SERIAL_NUMBER);

SLSHandle = SLS_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,
            OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL |
            FILE_FLAG_OVERLAPPED |SLS_OPEN_BY_SERIAL_NUMBER,0);

if (SLSHandle == INVALID_HANDLE_VALUE)
; // SLS_W32_CreateDevice failed
```

SLS_W32_Close Handle()

Protocol: `BOOL SLS_W32_CloseHandle (SLS_HANDLE SLSHandle)`

Description: Closes the specified device.

Parameters: *SLSHandle.* Handle of the device.

Return Value: If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero.

Example 1: The Example2-5. shows how to close a device after opening it for non-overlapped I/O using its description.

Example 2-5. Closing a Device after Opening

```
SLS_STATUS SLSStatus;
SLS_HANDLE SLSHandle;
char Buf[64];

SLSStatus = SLS_ListDevices(0,Buf,SLS_LIST_BY_INDEX
                            |SLS_OPEN_BY_DESCRIPTION);

SLSHandle = SLS_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,
                           0,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL
                           | SLS_OPEN_BY_DESCRIPTION,0);

if (SLSHandle == INVALID_HANDLE_VALUE)
 // SLS_W32_CreateDevice failed
else
{
 // SLS_W32_CreateFile OK, so do some work, and eventually ...

 SLS_W32_CloseHandle(SLSHandle);
}
```

SLS_W32_Read File()

Protocol: BOOL SLS_W32_ReadFile (SLS_HANDLE *SLSHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*, LPDWORD *lpdwBytesReturned*, PSLS_OVERLAPPED *lpOverlapped*)

Description: Reads data from the device.

Parameters:

SLSHandle. Handle of the device.

lpBuffer. Pointer to a buffer that receives the data from the device.

dwBytesToRead. Number of bytes to be read from the device.

lpdwBytesReturned. Pointer to a variable that receives the number of bytes read from the device.

LpOverlapped. Pointer to an sls_overlapped structure.

Return Value: If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero.

Remarks: This function supports both non-overlapped and overlapped I/O.

Non-overlapped I/O. The parameter, *lpOverlapped*, must be NULL for non-overlapped I/O.

- This function always returns the number of bytes read in *lpdw-BytesReturned*.
- This function does not return until *dwBytesToRead* have been read into the buffer.
- An application should use the function return value and *lpdwBytes-Returned* when processing the buffer. If the return value is non-zero and *lpdwBytesReturned* is equal to *dwBytesToRead* then the function has completed normally.
- A return value of SLS_IO_ERROR suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

Overlapped I/O. – When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

- The parameter, *lpOverlapped*, must point to an initialized OVER-LAPPED structure.

– If there is enough data in the receive queue to satisfy the request, the request completes immediately and the return code is non-zero. The number of bytes read is returned in *lpdwBytesReturned*.
– If there is not enough data in the receive queue to satisfy the request, the request completes immediately, and the return code is zero, signifying an error. An application should call `GetLastError` to get the cause of the error. If the error code is `ERROR_IO_PENDING`, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling `SLS_W32_GetOverlappedResult`.
– If successful, the number of bytes read is returned in *lpdwBytesReturned*.

Example 1: This example shows how to read 256 bytes from the device using non-overlapped I/O.

Example 2-6. Reading 256 bytes from the Device using non-overlapped I/O

```
SLS_HANDLE SLSHandle; /* setup by SLS_W32_CreateFile for
                         non-overlapped i/o*/
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;

if (SLS_W32_ReadFile(SLSHandle,Buf,dwToRead,&dwRead,&osWrite))
{
  if (dwToRead == dwRead)
    ; // SLS_W32_ReadFile OK
  else
    ; // SLS_W32_ReadFile timeout
}
else
   ; // SLS_W32_ReadFile failed
```

Example 1: The Example2-7. shows how to read 256 bytes from the device using *overlapped I/O*.

Example 2-7. Reading 256 bytes from the Device using Overlapped I/O

```
SLS_HANDLE SLSHandle; // setup by SLS_W32_CreateFile for
overlapped i/o
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;
SLS_OVERLAPPED osRead;

//Initialize sls overlapped structure
osRead.Offset =0;
osRead.OffsetHigh =0;
osRead.hEvent =CreateEvent(NULL,false,false,NULL);

if (!SLS_W32_ReadFile(SLSHandle,Buf,dwToRead,&dwRead,&osWrite))
{
  int iStatus = GetLastError();
  if(iStatus ==ERROR_IO_PENDING)
  {
     DWORD dwResult = WaitForSingleObject(ov.hEvent,INFINITE);
     switch(dwResult)
     {
        case WAIT_ABANDONED:
        // wait abandoned
        case WAIT_OBJECT_0:
        {
           if(SLS_W32_GetOverlappedResult(hDevice,&ov,&junk,TRUE))
           {
              if (dwToRead == dwRead)
              // SLS_W32_ReadFile OK
              else
              // SLS_W32_ReadFile timeout
           }
        }
        case WAIT_TIMEOUT:
        // time out
     }
  }
  else
  {
     // SLS_W32_ReadFile OK
  }
}
```

SLS_W32_Write File()

Protocol: BOOL SLS_W32_WriteFile (SLS_HANDLE *SLSHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*, PSLS_OVERLAPPED *lpOverlapped*)

Description: Writes data to the device.

Parameters:

SLSHandle. Handle of the device.

lpBuffer. Pointer to the buffer that contains the data to write to the device.

dwBytesToWrite. Number of bytes to be written to the device.

lpdwBytesWritten. Pointer to a variable that receives the number of bytes written to the device.

lpOverlapped. Pointer to an sls_overlapped structure.

Return Value: If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero.

Remarks: This function supports both non-overlapped and overlapped I/O.

Non-overlapped I/O. The parameter, *lpOverlapped*, must be NULL for non-overlapped I/O.

- This function always returns the number of bytes written in *lpdwBytesWritten*.
- This function does not return until *dwBytesToWrite* have been written to the device.
- An application should always use the function return value and *lpdwBytesWritten*. If the return value is non-zero and *lpdwBytesWritten* is equal to *dwBytesToWrite* then the function has completed normally.

Overlapped I/O. When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

- The parameter, *lpOverlapped*, must point to an initialized SLS_OVERLAPPED structure.

– This function completes immediately, and the return code is zero, signifying an error. An application should call `GetLastError` to get the cause of the error. If the error code is `ERROR_IO_PENDING`, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling `SLS_W32_GetOverlappedResult`.

– If successful, the number of bytes written is returned in *`lpdwBytesWritten`*.

Example 1: The Example2-8. shows how to write 128 bytes to the device using non-overlapped I/O.

Example 2-8. Writing 128 bytes to the Device using Nonoverlapped I/O

```
SLS_HANDLE SLSHandle; // setup by SLS_W32_CreateFile for overlapped
i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;

if (SLS_W32_WriteFile(SLSHandle,Buf,dwToWrite,&dwWritten,&osWrite))
{
   if (dwToWrite == dwWritten)
     // SLS_W32_WriteFile OK
   else
     // SLS_W32_WriteFile timeout
}
else
  // SLS_W32_WriteFile failed
```

Example 2: The Example2-9. shows how to write 128 bytes to the device using overlapped I/O.

Example 2-9. Writing 128 bytes to the Device using Overlapped I/O

```
SLS_HANDLE SLSHandle; // setup by SLS_W32_CreateFile for
overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;
SLS_OVERLLAPED osWrite;

//Initialize sls overlapped structure
osWrire.Offset =0;
osWrite.OffsetHigh =0;
osWrite.hEvent =CreateEvent(NULL,false,false,NULL);

if
(!SLS_W32_WriteFile(SLSHandle,Buf,dwToWrite,&dwWritten,&osWrite))
{
   DWORD ErrorCode = GetLastError();
   if (ErrorCode == ERROR_IO_PENDING)
   {
      // write is delayed so do some other stuff until ...
     DWORD dwResult = WaitForSingleObject(osWrite.hEvent,INFINITE);
      switch(dwResult)
      {
         case WAIT_ABANDONED:
            // wait abandoned
         case WAIT_OBJECT_0:
         {
            if(!SLS_W32_GetOverlappedResult(SLSHandle,&osWrite,
                                            &dwWritten,FALSE))
               // error
            else
            {
               if (dwToWrite == dwWritten)
                  // SLS_W32_WriteFile OK
               else
                  // SLS_W32_WriteFile timeout
            }
         }
      }
   }
}
else
{
   // SLS_W32_WriteFIle OK
}
```

SLS_W32_GetOverlappedResult()

Protocol: BOOL `SLS_W32_GetOverlappedResult` (`SLS_HANDLE` *`SLSHandle`*, `PSLS_OVERLAPPED` *`lpOverlapped`*, `LPDWORD` *`lpdwBytesTransferred`*, `BOOL` *`bWait`*)

Description: Gets the result of an overlapped operation.

Parameters:

`SLSHandle.` Handle of the device.

`lpOverlapped.` Pointer to an overlapped structure.

`lpdwBytesTransferred.` Pointer to a variable that receives the number of bytes transferred during the overlapped operation.

`bWait.` Set to TRUE if the function does not return until the operation has been completed.

Return Value: If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero.

Remarks: This function is used with overlapped I/O. For description of its use, see "SLS_W32_Read File()" on page 10 and "SLS_W32_Write File()" on page 13

SLS_Reset Device()

Protocol: `SLS_STATUS SLS_ResetDevice (SLS_HANDLE slsHandle)`

Description: This function sends a reset command to the device.

Parameters: *slsHandle.* Handle of the device to reset.

Return Value: `SLS_OK` if successful, otherwise the return value is an SLS error code.