SIS3153
USB Windows API Reference

SIS GmbH
Harksheider Str. 102A
22399 Hamburg
Germany

Phone: ++49 (0) 40 60 87 305 0
Fax: ++49 (0) 40 60 87 305 20

email: info@struck.de
http://www.struck.de

Version: sis3153-M-usb-3-v101-windows_Api_Reference.doc as of 05.06.2019

Revision Table:

Revision	Date	Modification
1.0	28.10.2013	First official release
1.1	05.06.2019	Add VME wrapper functions

- Table of contents

1 Overview

This section provides an overview over the implemented SIS3153API functions.
See `sis3153usb_vme_class.cpp` and `sis3153usb_vme_class.h`

API Function Name	Description
FindAll_SIS3153USB_Devices	Find amount of connected devices
Sis3153usb_OpenDriver	Get device handle based on device index
Sis3153usb_CloseDriver	Close previously opened device handle
sis3153Usb_version	Get driver version
sis3153Usb_Register_Single_Read	Read single data from internal control register space
sis3153Usb_Register_Single_Write	Write single data to internal control register space
sis3153Usb_Register_Dma_Read	Read data blocks from internal control register space
sis3153Usb_Register_Dma_Write	Write data blocks to internal control register space
sis3153Usb_Vme_Single_Read	Read single cycle from VME address space
sis3153Usb_Vme_Single_Write	Write single cycle to VME address space
sis3153Usb_Vme_Dma_Read	Read blocks from VME address space
sis3153Usb_Vme_Dma_Write	Write blocks to VME address space
sis3153Usb_VmeSysreset	Issue VME System reset to crate
Additional VME wrapper functions	**Several functions for access VME address space**
vme_CRCSR_D*_read	
vme_CRCSR_D*_write	
…	
vme_A8D*_read	
vme_A8D*_write	
…	
vme_A16D*_read	
vme_A16D*_write	
…	
vme_A32D*_read	
vme_A32D*_write	
…	

1.1 FindAll_SIS3153USB_Devices

Syntax:

```
SIS3153W_STATUS
FindAll_SIS3153USB_Devices(
            struct SIS3153USB_Device_Struct* sis3153usb_Device,
            UINT *nof_usbdevices,
            UINT max_usb_device_Number
);
```

Description:
Returns the amount of connected SIS3153 cards and its "sis3153usb_Device" informations (`SIS3153USB_Device_Struct`).

Arguments:

sis3153usb_Device
: Pointer to a device handle structure.

nof_usbdevices
: Pointer to unsigned 32bit value which will hold the amount of found devices

max_usb_device_Number
: Maximum number of devices which can be managed by the driver at the same time.

Return Codes:

Code	Description
`Stat3153Success`	The function returned successfully
`Stat3153NullArgument`	One or more pointer arguments is/are NULL

Usage:

```
#define MAX_USB_DEV_NUMBER 4
SIS3153W_STATUS   status;
struct SIS3153USB_Device_Struct sis3153_device[MAX_USB_DEV_NUMBER];
UINT gl_nof_usbdevices;


status = FindAll_SIS3153USB_Devices(
                  sis3153_device,
                  &gl_nof_usbdevices,
                  MAX_USB_DEV_NUMBER
);


if(status != Stat3153Success){
      printf("Error in FindAll_SIS3153USB_Devices': %x\n", status);
}
else{
      printf("found %d device(s)\n", found);
}
```

1.2 Sis3153usb_OpenDriver

Syntax:

```
SIS3153W_STATUS
Sis3153usb_OpenDriver(
            struct SIS3153USB_Device_Struct* sis3153usb_Device
);
```

Description:
Tries to open a device handle for a card with the supplied device handle structure pointer.

Arguments:

sis3153usb_Device

Pointer to a device handle structure.

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL
Stat3153OpenFailedWithIndex	Failed to open device on supplied index

Usage:

```
SIS3153W_STATUS   status;
struct SIS3153USB_Device_Struct sis3153_device


status = Sis3153usb_OpenDriver(sis3153_device);


if(status != Stat3153Success){
      printf("Error in Sis3153usb_OpenDriver: %x\n", status);
}
```

1.3 sis3153Usb_version

Syntax:

```
SIS3153W_STATUS
sis3153Usb_version(
            PSIS3153W_VERSION ver
);
```

Description:
Returns revision information of the used API.

Arguments:

ver

Pointer to a SIS3153W_VERSION structure. See type explanations in appendix.

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153W_STATUS status;
SIS3153W_VERSION ver;

status = sis3153Usb_version(&ver);

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_version': %x\n", status);
}else{
      printf("API Version: v%d.%d\n", ver.major, ver.minor);
}
```

1.4 sis3153usb_CloseDriver

Syntax:

```
SIS3153W_STATUS
sis3153usb_CloseDriver(
	PSIS3153_DEV_STRUCT stDev
	);
```

Description:
Closes a previously opened device handle.

Arguments:
stDev
Pointer to a device handle structure.

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153_DEV_STRUCT stDev; // previously opnened
SIS3153W_STATUS status;

status = sis3153usb_Close(stDev);

if(status != Stat3153Success){
	printf("Error in 'sis3153usb_CloseDriver': %x\n", status);
}
```

1.5 sis3153Usb_Register_Single_Read

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Register_Single_Read(
      HANDLE usbDevice,
      UINT   addr,
      UINT   *data
      );
```

Description:
Reads from the USB modules internal control register space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: register offset to read from

data
: Pointer to an unsigned 32bit variable to hold the read data

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153W_STATUS status;
SIS3153_DEV_STRUCT hDev; // previously opened
UINT data;

status = sis3153Usb_Register_Single_Read(
                        &hUsb,
                        0x0,
                        &data
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Register_Single_Read': %x\n", status);
}
```

1.6 sis3153Usb_Register_Single_Write

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Register_Single_Write(
      HANDLE usbDevice,
      UINT addr,
      UINT data
      );
```

Description:
Writes to the USB modules internal register space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: register offset to write to

data
: Unsigned 32bit variable which holds the data to write

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153W_STATUS status;
SIS3153_DEV_STRUCT hDev; // previously opened

status = sis3153Usb_Register_Single_Write(
                        &hUsb,
                        0x0,
                        0x12345678
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Register_Single_Write: %x\n", status);
}
```

1.7 sis3153Usb_Register_Dma_Read

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Register_Dma_Read(
	HANDLE usbDevice,
	UINT addr,
	UINT *dmabufs,
	UINT req_nof_data,
	UINT *got_nof_data
);
```

Description:
Reads, using block transfer access, from internal register space of the USB module.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: register offset to read from

dmabufs
: Pointer to an unsigned 32bit buffer which holds the data

req_nof_data
: Requested number of 32bit word to read

got_nof_data
: Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153W_STATUS status;
SIS3153_DEV_STRUCT hDev; // previously opened
UINT data[1000];
UINT numGot;


status = sis3153Usb_Register_Dma_Read(
                        &hUsb,
                        0x123,
                        data,
                        1000,
                        &numGot
                        );

if(status != Stat3153Success){
	printf("Error in 'sis3153Usb_Register_Dma_Read: %x\n", status);
}
```

1.8 sis3153Usb_Register_Dma_Write

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Register_Dma_Write(
	HANDLE usbDevice,
	UINT addr,
	UINT *dmabufs,
	UINT req_nof_data,
	UINT *put_nof_data
);
```

Description:
Writes, using block transfer access, to internal register space of the USB module.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: register offset to write to

dmabufs
: Pointer to an unsigned 32bit buffer which holds the data

req_nof_data
: Requested number of 32bit word to write

put_nof_data
: Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL

Usage:

```
SIS3153W_STATUS status;
SIS3153_DEV_STRUCT hDev; // previously opened
UINT data[1000];
UINT numPut;

status = sis3153Usb_Register_Dma_Write(
                        &hUsb,
                        0x123,
                        data,
                        1000,
                        &numPut
                        );

if(status != Stat3153Success){
	printf("Error in 'sis3153Usb_Register_Dma_Write: %x\n", status);
}
```

1.9 sis3153Usb_Vme_Single_Read

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Vme_Single_Read(
       HANDLE usbDevice,
       UINT addr,
       UINT am,
       UINT size,
       UINT *data
       );
```

Description:
Reads, using single cycle access from VME address space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: VME address offset

am
: VME address modifier

size
: Transfer size (1, 2 or 4 bytes)

data
: Pointer to an unsigned 32bit value to hold the read data

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL
Berr...	VME Bus error

Usage:

```
SIS3153W_STATUS status;
UINT data;

status = sis3153Usb_Vme_Single_Read(
                        &hUsb,
                        0x0, // VME address 0
                        0x9, // AM (A32)
                        0x4, // D32
                        &data
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Vme_Single_Read': %x\n", status);
}
```

1.10 sis3153Usb_Vme_Single_Write

Syntax:

```
SIS3153W_STATUS
sis3153Usb_VmeSingleWrite(
       HANDLE usbDevice,
       UINT addr,
       UINT am,
       UINT size,
       UINT data
       );
```

Description:
Writes, using single cycle access to VME address space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: VME address offset

am
: VME address modifier

size
: Transfer size (1, 2 or 4 bytes)

data
: Unsigned 32bit value which holds the data to write

Return Codes:

Code	Description
Stat3153Success	The function returned successfully
Stat3153NullArgument	At least 1 pointer argument is NULL
Berr...	VME Bus error

Usage:

```
SIS3153W_STATUS status;
UINT data = 0xDEADBEEF;

status = sis3153Usb_Vme_Single_Write(
                        &hUsb,
                        0x0, // VME address 0
                        0x9, // AM (A32)
                        0x4, // D32
                        data
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Vme_Single_Write': %x\n", status);
}
```

1.11 sis3153Usb_Vme_Dma_Read

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Vme_Dma_Read(
       HANDLE usbDevice,
       UINT   addr,
       UINT   am,
       UINT   size,
       UINT   fifo_mode,
       UINT   *dmabufs,
       UINT   req_nof_data,
       UINT   *got_nof_data
       );
```

Description:
Reads, using block transfer access from VME address space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: VME address offset

am
: VME address modifier

size
: Transfer size (1, 2 or 4 bytes)

fifo_mode
: Keep VME address static

dmabufs
: Pointer to an unsigned 32bit buffer to hold the read data

req_nof_data
: Requested number of 32bit word to read

got_nof_data
: Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
`Stat3153Success`	The function returned successfully
`Stat3153NullArgument`	At least 1 pointer argument is NULL
`Berr...`	VME B us error

Usage:

```
SIS3153W_STATUS status;
UINT data[1000];
UINT numGot;

status = sis3153Usb_Vme_Dma_Read(
                        &hUsb,
                        0x0, // VME address 0
                        0x9, // AM (A32)
                        0x4, // D32
                        0x0, // no fifo mode
                        data,
                        1000,
                        &numGot
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Vme_Dma_Read': %x\n", status);
}
```

1.12 sis3153Usb_Vme_Dma_Write

Syntax:

```
SIS3153W_STATUS
sis3153Usb_Vme_Dma_Write(
       HANDLE usbDevice,
       UINT   addr,
       UINT   am,
       UINT   size,
       UINT   fifo_mode,
       UINT   *dmabufs,
       UINT   req_nof_data,
       UINT   *put_nof_data
       );
```

Description:
Writes, using block transfer access to VME address space.

Arguments:

usbDevice
: Pointer to an opened device.

addr
: VME address offset

am
: VME address modifier

size
: Transfer size (1, 2 or 4 bytes)

fifo_mode
: Keep VME address static

dmabufs
: Pointer to an unsigned 32bit buffer which holds the data

req_nof_data
: Requested number of 32bit word to write

put_nof_data
: Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
`Stat3153Success`	The function returned successfully
`Stat3153NullArgument`	At least 1 pointer argument is NULL
`Berr...`	VME Bus error

Usage:

```
SIS3153W_STATUS status;
UINT data[1000];
UINT numPut;

status = sis3153Usb_Vme_Dma_Write(
                        &hUsb,
                        0x0, // VME address 0
                        0x9, // AM (A32)
                        0x4, // D32
                        0x0, // no fifo mode
                        data,
                        1000,
                        &numPut
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_Vme_Dma_Write': %x\n", status);
}
```

1.13 sis3153Usb_VmeSysreset

Syntax:

```
SIS3153W_STATUS
sis3153Usb_VmeSysreset(
      HANDLE usbDevice
      );
```

Description:
Issues a 300ms Sysreset to the VME crate.

Arguments:
`usbDevice`
Pointer to an opened device.

Return Codes:

Code	Description
`Stat3153Success`	The function returned successfully
`Stat3153NullArgument`	At least 1 pointer argument is NULL

Usage:

```
SIS1100W_STATUS status;

status = sis3153Usb_VmeSysreset(
                        usbDevice
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153Usb_VmeSysreset': %x\n", status);
}
```

2 Additional VME wrapper functions methods

Based on the main VME read/write functions (single/DMA) several VME wrapper functions are provided. These functions allow access to the VME bus with standardised setup for Mode and Size:

Example with “vme_A16D16_write”:

```
unsigned int vme_address = 0x3800 ; // A16 Address
return_code = vme_A16D16_write(vme_crate, vme_address, ushort_data); //
```

return_codes:
0x211: VME Buss Error
0x212: VME Retry
0x214: VME Arbitration Timeout

2.1 VME read functions/methods

```
int vme_CRCSR_D8_read(HANDLE hXDev, UINT vme_adr, UCHAR* vme_data);
int vme_CRCSR_D16_read(HANDLE hXDev, UINT vme_adr, USHORT* vme_data);
int vme_CRCSR_D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data);

int vme_A16D8_read(HANDLE hXDev, UINT vme_adr, UCHAR* vme_data);
int vme_A16D16_read(HANDLE hXDev, UINT vme_adr, USHORT* vme_data);
int vme_A16D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data);

int vme_A24D8_read(HANDLE hXDev, UINT vme_adr, UCHAR* vme_data);
int vme_A24D16_read(HANDLE hXDev, UINT vme_adr, USHORT* vme_data);
int vme_A24D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data);
int vme_A24DMA_D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A24BLT32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A24MBLT64_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A24DMA_D32FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A24BLT32FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A24MBLT64FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
```

```
int vme_A32D8_read(HANDLE hXDev, UINT vme_adr, UCHAR* vme_data);
int vme_A32D16_read(HANDLE hXDev, UINT vme_adr, USHORT* vme_data);
int vme_A32D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data);
int vme_A32DMA_D32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A32BLT32_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A32MBLT64_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A32DMA_D32FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_no_of_lwords);
int vme_A32BLT32FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32MBLT64FIFO_read(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);

int vme_A32_2EVME_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2EVMEFIFO_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);

int vme_A32_2ESST160_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2ESST160FIFO_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2ESST267_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2ESST267FIFO_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2ESST320_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
int vme_A32_2ESST320FIFO_read(HANDLE  hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* got_num_of_lwords);
```

2.2 VME write functions/methods:

```
int vme_CRCSR_D8_write(HANDLE hXDev, UINT vme_adr, UCHAR vme_data);
int vme_CRCSR_D16_write(HANDLE hXDev, UINT vme_adr, USHORT vme_data);
int vme_CRCSR_D32_write(HANDLE hXDev, UINT vme_adr, UINT vme_data);

int vme_A16D8_write(HANDLE hXDev, UINT vme_adr, UCHAR vme_data);
int vme_A16D16_write(HANDLE hXDev, UINT vme_adr, USHORT vme_data);
int vme_A16D32_write(HANDLE hXDev, UINT vme_adr, UINT vme_data);

int vme_A24D8_write(HANDLE hXDev, UINT vme_adr, UCHAR vme_data);
int vme_A24D16_write(HANDLE hXDev, UINT vme_adr, USHORT vme_data);
int vme_A24D32_write(HANDLE hXDev, UINT vme_adr, UINT vme_data);
int vme_A24DMA_D32_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A24BLT32_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A24MBLT64_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A24DMA_D32FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A24BLT32FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A24MBLT64FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);

int vme_A32D8_write(HANDLE hXDev, UINT vme_adr, UCHAR vme_data);
int vme_A32D16_write(HANDLE hXDev, UINT vme_adr, USHORT vme_data);
int vme_A32D32_write(HANDLE hXDev, UINT vme_adr, UINT vme_data);
int vme_A32DMA_D32_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32BLT32_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32MBLT64_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32DMA_D32FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32BLT32FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32MBLT64FIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);

int vme_A32_2EVME_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords);
int vme_A32_2EVMEFIFO_write(HANDLE hXDev, UINT vme_adr, UINT* vme_data, UINT req_num_of_lwords, UINT* put_num_of_lwords)
```

3 Data Structures

This section provides an overview over the available data structures.
See sis3153wType.h and/or sis3153wStat.h for reference.

3.1 SIS3153USB_Device_Struct

Syntax:

```
struct SIS3153USB_Device_Struct{
      PVOID hDev;
      PVOID hUsb;
      PCHAR cDName;
      PCHAR readableName;
      USHORT idVendor;
      USHORT idProduct;
      USHORT idSerNo;
      USHORT idDriverVersion;
      USHORT idFxFirmwareVersion;
      USHORT idFpgaFirmwareVersion;
};
```

Description:
Userspace card handle to be able to handle more than one card simultaneously.
This structure needs to be passed to all function which will access the hardware.
The members of this type should never be written directly.

3.2 SIS3153W_VERSION

Syntax:

```
// version information
typedef struct _SIS3153W_VERSION{
      UCHAR SisMajor;
      UCHAR SisMinor;
}SIS3153W_VERSION, *PSIS3153W_VERSION;
```

Description:
Struct used to retrieve API revision information via the sis3153Usb_version call.

3.3 SIS3153W_STATUS

Syntax:

```
#define API_RETURNCODE_START 0x0
#define API_RETURNCODE_LOCAL 0x100
#define API_RETURNCODE_REMOTE 0x200

typedef enum _SIS3153W_STATUS{
      // 0: general return codes
      Stat3153Success = API_RETURNCODE_START,
      Stat3153NullArgument,
      Stat3153InvalidDeviceIndex,
      Stat3153InvalidArgument,
      Stat3153ErrorDeviceOpen,
      Stat3153ErrorDeviceHold,
      Stat3153ErrorDeviceDownload,
      Stat3153ErrorDeviceRun,
      Stat3153ErrorDeviceClose,
      Stat3153ErrorSetInterface,
      Stat3153ErrorDescriptorOpen,
      Stat3153ErrorIo,
      Stat3153ErrorFileOpen,
      Stat3153ErrorFileAlloc,
      Stat3153ErrorPromId,
      Stat3153ErrorPromTimeout,
      Stat3153ErrorVfyAlloc,
      Stat3153ErrorVfyFailed,
      // 0x100: local side errors
      Stat3153InvalidParameter = API_RETURNCODE_LOCAL + 0x10,
      Stat3153UsbWriteError,
      Stat3153UsbReadError,
      Stat3153UsbReadLengthError,
      // 0x200: remote side errors
      Stat3153VmeBerr = API_RETURNCODE_REMOTE + 0x11 // starts at 0x211
}SIS3153W_STATUS;
```

Description:
A enum type to have a readable representation of the possible return code from each API function.

4 Index