Programmers Guide

for

INFINITY® USB SMART

SDK 1.00

Introduction

The Infinity® USB Smart SDK provides a simple programming interface for the Infinity USB Smart. The SDK makes it easy to access the ISO7816 (phoenix), SLE and I2C memory interface of the Infinity USB Smart.

System overview

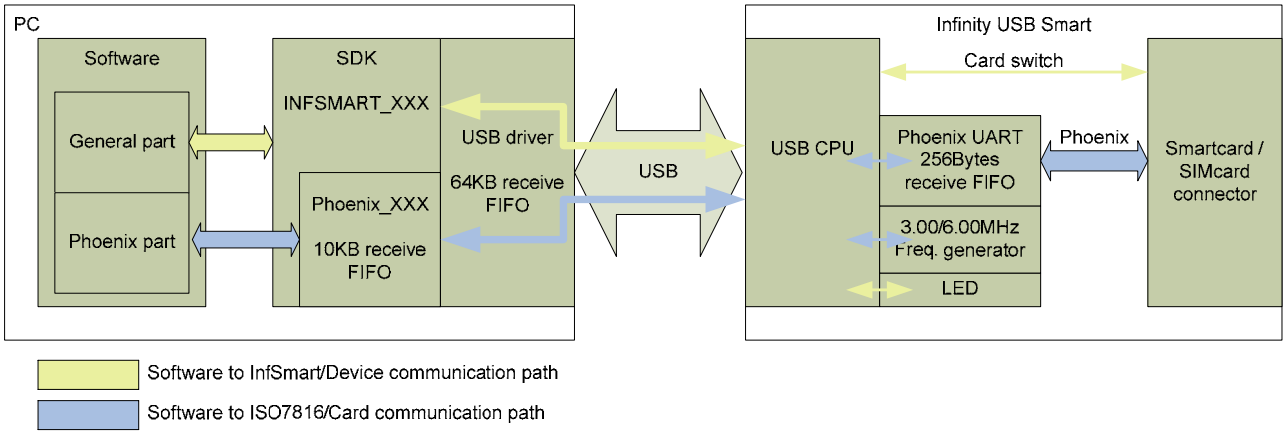

Figure 1. System overview

Typical software that uses the SDK contains a general part that handles finding (enumerate), opening and initialization of the device and another part that handles the phoenix/SLE or I2C communications.

The general part should use the functions named INFSMART_XXX to find, open and setup the device. The software should then use the INFSMART_Phoenix_XXX functions to setup the UART of the device and communicate with the card through the phoenix interface or use the I2C or SLE functions.

The phoenix UART consists of a double-buffered 256 byte receive buffer to handle incoming data from the card. The phoenix part of the SDK needs to empty this buffer before overflow occurs and this is done by using the INFSMART_Phoenix_Read or INFSMART_Phoenix_BytesInFifo. The BytesInFifo function, copies data from the device to the internal 10KB receive FIFO (located in the SDK-DLL) and then returns the total amount of data available. If 256 bytes FIFO is not enough you should call INFSMART_Phoenix_Read continuously to empty the device fifo into its internal 10KB FIFO.

Communication

In general, the user initiates communication with the Infinity USB Smart by making a call to INFSMART_GetNumDevices. This call will return the number of devices connected and is used as a range when calling INFSMART_OpenDeviceFromNum. The handle returned from INFSMART_OpenDeviceFromNum should be used in all subsequent calls to the SDK. Once the device is opened phoenix mode can be enabled using INFSMART_Phoenix_Enable.

Data I/O through phoenix mode is performed using INFSMART_Phoenix_Write and INFSMART_Phoenix_Read functions. When phoenix mode operation is complete, phoenix mode is disabled by a call to INFSMART_Phoenix_Disable. When all I/O operations are complete, the device is closed by a call to INFSMART_Close.

Supporting multiple devices

This SDK supports multiple Infinity USB Smart connected simultaneously. Making support for multiple programmers in your software is more advanced than supporting only one device. Before you begin using this SDK you should consider if you need to support multiple devices simultaneously. If you wish to support only one device (which is only needed in most cases), you would normally use a 0 for the parameter dwDevice in INFSMART_OpenDeviceFromNum, which refers to the first connected device. If another instance of your application is opened you could make support for multiple devices by not always using device 0, but instead use the first available closed device.

SDK functions

The SDK is divided into 5 categories:

- **General**
  The general functions are used to find, open, and close devices, handle timeout values of read and write operations, to set the state of the LED, and detect if any card is inserted.

- **Phoenix**
  The phoenix functions are used to control, read and write the ISO7816 interface. These functions are designed to make it easy for developers of other serial-based phoenix software to easily adapt their software to use this SDK instead.

  For writing a phoenix enabled application only the above 2 categories are needed, the next category, communications, are for more advanced users familiar with the communication protocol of the Infinity USB Smart.

- **Communications**
  These functions are used to read and write data directly to the Infinity USB Smart, to flush the data buffer and to determine how many bytes are available in the read buffer.

- **Synchronous communications**
  These functions are used for accessing SLE cards which use synchronous IO.

- **I2C communications**
  These functions are used for accessing I2C EEproms directly.

General

INFSMART_GetSDKVersion

Returns the version of the used SDK.

Prototype:
```
DWORD INFSMART_GetSDKVersion();
```

Parameters:
-None

Return values:
A DWORD representing the version of the used SDK. For instance 2100 for version 2.1.

INFSMART_GetNumDevices

This function returns the number of Infinity USB Smart currently connected to the PC. If 1 device is connected lpdwNumDevices will contain the value 1 on return.

Prototype:
```
SDK_STATUS INFSMART_GetNumDevices(LPDWORD lpdwNumDevices);
```

Parameters:
1. lpdwNumDevices: Address of a DWORD variable that will contain the number of devices connected on return.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_NOT_FOUND (0xFF) or
SDK_INVALID_PARAMETER (0x06)

INFSMART_OpenDeviceFromNum

This function opens the specified device, using a zero-indexed device number as returned by INFSMART_GetNumDevices. If 1 device is connected you should specify 0 as dwDevice.

Prototype:
SDK_STATUS INFSMART_OpenDeviceFromNum(DWORD dwDevice, HANDLE* hDevice, LPVOID lpDeviceString);

Parameters:
1. dwDevice: Zero-based index of the device to open. Use INFSMART_GetNumDevices to find number of connected devices.
2. hDevice: Address of a handle which will receive the handle of the opened device, later to be used in subsequent calls to the device.
3. lpDeviceString: Address of a buffer to be filled with a unique hardware string identifying the unique instance of the connected device, should be large enough to contain 256 bytes. Set to 0 if you do not wish to receive the device string.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_NOT_FOUND (0xFF) or
SDK_INVALID_PARAMETER (0x06)

INFSMART_CloseDevice

This function closes the specified device which has previously been opened by INFSMART_OpenDeviceFromNum.

Prototype:
SDK_STATUS INFSMART_CloseDevice(HANDLE hDevice);

Parameters:
1. hDevice: Handle of the device to close. This handle should be a handle previously received from INFSMART_OpenDeviceFromNum.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_PARAMETER (0x06)

INFSMART_SetLEDState

Sets the LED to the specified color, brightness and flash pattern (PWM).

Prototype:
SDK_STATUS INFSMART_SetLEDState(HANDLE hDevice, unsigned int R, unsigned int G,unsigned int B, unsigned char PWMDuty, unsigned char PWMFrq);

Parameters:

1. hDevice: Handle of the device.
2. R: 16bit value specifying the amount of RED. (0x0000 = Off, 0xFFFF = On)
3. G: 16bit value specifying the amount of GREEN. (0x0000 = Off, 0xFFFF = On)
4. B: N/A
5. PWMDuty: 8bit value specifying the on/off duty cycle of the flash. (0x01 = mostly off, 0xFE = Mostly on, 0x80 = (50% off, 50% on))
6. PWMFrq: 8bit value specifying the frequency of the flash. (0x00 = Slow, 0xFF = Fast)

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_WRITE_ERROR (0x04)

INFSMART_GetCardState

Returns the state of the card switches of both the smartcard and SIM-card connector.

Prototype:
SDK_STATUS INFSMART_GetCardState(HANDLE hDevice, LPDWORD lpdwCardstate);

Parameters:
1. hDevice: Handle of the device.
2. lpdwCardstate: Address of a DWORD which will contain the state of the card switches.

   The predefined values are:
   0 – No cards inserted (CARD_NONE)
   1 – Smartcard inserted (CARD_SMARTCARD)
   2 – SIM-card inserted (CARD_SIMCARD)
   3 – Both cards inserted (CARD_BOTH)

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01)

INFSMART_GetFirmwareVersion

Returns the version of the firmware in the programmer.

Prototype:
DWORD INFSMART_GetFirmwareVersion(HANDLE hDevice);
Parameters:
1. hDevice: Handle of the device.

Return values:
A DWORD representing the version of the firmware in the programmer. A version of 1030 indicates firmware version 1.03.

PHOENIX

INFSMART_Phoenix_Enable

Enables phoenix mode and sets the uart to the specified baud-rate and parity. This function should be called before using any of the INFSMART_Phoenix functions.

Prototype:
SDK_STATUS INFSMART_Phoenix_Enable(HANDLE hDevice, unsigned int baud, unsigned int* ActualBaud, unsigned char stopbit, unsigned char parity, unsigned char databits, unsigned int iFrq, unsigned int iCardFrq, bool updateonly);

Parameters:

1. hDevice: Handle of the device.
2. baud: Value indicating which baudrate the card uses at the specific frequency.
3. ActualBaud: Pointer to an integer to receive the actual baudrate, the actual baudrate may deviate from the requested.
4. stopbit: Value indication the use of stopbit.

   0x00 – Short stopbit
   0x01 – Long stopbit

5. parity:

   0x00 – No parity
   0x01 – Odd parity
   0x02 – Even parity
   0x03 – Mark parity
   0x04 – Space parity

6. databits: Specifies the number of databits to use. 5 to 8 databits are valid.
7. iFrq: The frequency the card expects, for instance 3579000Hz or 6000000Hz.
8. iCardFrq: The actual frequency used
9. updateonly: Specified whether just the baud-rate or other uart settings need to be updated, or if phoenix mode needs to be enabled.

   false : Default mode, enables phoenix mode with the requested values
   true : Just updates the UART with the new baudrate, stopbit and parity

ISO7816 normally uses 9600bps with EVEN parity and two stopbits.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06)

INFSMART_Phoenix_Disable

Disables phoenix mode and returns to normal state.

Prototype:
SDK_STATUS INFSMART_Phoenix_Disable(HANDLE hDevice);

Parameters:
1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_Write

This function uses INFSMART_Write to write data to the device that is redirected to the ISO7816 (phoenix) interface. Data is sent through the hardware UART with the baudrate and properties specified in the INFSMART_Phoenix_Enable function. Phoenix mode has to be enabled using INFSMART_Phoenix_Enable prior to using this function. The device has a 256 bytes phoenix receive buffer, so do not request more data than 256 bytes before you empty the read buffer. Each byte transmitted will also be echoed. For example transmitting 128bytes to the card, which replyes with a 2byte ACK will result in 130bytes of data to be read.

The function writes the specified number of bytes from the specified buffer to the specified device. Given valid parameters, this function is blocking until the write is successful, fails, or a timeout occurs. The write is successful when the device has accepted all of the data. If the write fails or a timeout occurs, SDK_WRITE_ERROR is returned.

Prototype:
SDK_STATUS INFSMART_Phoenix_Write(HANDLE hDevice, LPCVOID lpBuffer,DWORD nNumberOfBytesToWrite, LPDWORD lpdwBytesWritten);

Parameters:
1. hDevice: Handle of the device.
2. lpBuffer: Address of a buffer of data to write.
3. nNumberOfBytesToWrite: Number of bytes to write to the device (0-4096 bytes)
4. lpdwBytesWritten: Address of a DWORD which will contain the number of bytes actually written to the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_WRITE_ERROR (0x04) or
SDK_INVALID_REQUEST_LENGTH (0x07) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_Read

This function uses INFSMART_Read and INFSMART_Write to read data from the device, which the device has accepted from the ISO7816 (phoenix) interface. Data is read through the hardware uart with the baud-rate and properties specified in the INFSMART_Phoenix_Enable function.
Phoenix mode has to be enabled using INFSMART_Phoenix_Enable prior to this function.
The device has a 256 bytes phoenix receive buffer, do not request more data than 256 bytes at a time from the ISO7816 interface, before you read the buffer. If 256 bytes are not enough to receive the requested data, make sure to call INFSMART_Phoenix_Read continuously as this will copy data from the device's receive buffer to the SDK's receive buffer which is 10KB.

The function reads the specified number of bytes into the specified buffer and retrieves the number of bytes read. Given valid input parameters, this function is always blocking until the specified number of bytes is available. Use INFSMART_Phoenix_BytesInFifo, prior to calling this function to make sure the requested bytes are actually ready to be read.

Prototype:
```
SDK_STATUS INFSMART_Phoenix_Read(HANDLE hDevice, LPVOID lpBuffer,DWORD
nNumberOfBytesToRead, LPDWORD lpdwBytesRead);
```

Parameters:
1. hDevice: Handle of the device.
2. lpBuffer: Address of a buffer to receive the data.
3. nNumberOfBytesToRead: Number of bytes to read from the device (0-64Kbytes)
4. lpdwBytesRead: Address of a DWORD which will contain the number of bytes actually read from the device.

Return values:
```
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_READ_ERROR (0x02) or
SDK_INVALID_REQUEST_LENGTH (0x07) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06) or
SDK_PHOENIX_NOT_ENABLED (0x60)
```

INFSMART_Phoenix_SetRST

Sets the RST (reset) pin of the card connector to the specified state.

Prototype:
SDK_STATUS INFSMART_Phoenix_SetRST(HANDLE hDevice, unsigned int state);

Parameters:
1. hDevice: Handle of the device.
2. state: Specifies the state of the reset pin, set to 1 to set the card in reset state, and 0 the set the card out of reset state.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_ResetCard

Flushes the phoenix receive buffer, then sets the RST (reset) pin state low (in-reset), waits, then sets the RST high again (out-reset). Under normal circumstances you should use the INFSMART_Phoenix_Read afterwards to read the ATR that usually follows a card-reset from the card. Use the INFSMART_Phoenix_BytesInFifo function to determine the size of the ATR to read.

Prototype:
```
SDK_STATUS INFSMART_Phoenix_ResetCard(HANDLE hDevice);
```

Parameters:
1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_BytesInFifo

Returns the number of bytes ready to be read from the phoenix read buffer.

Prototype:
```
DWORD INFSMART_Phoenix_BytesInFifo(HANDLE hDevice);
```

Parameters:
1. hDevice: Handle of the device.

Return values:
Number of bytes ready to be read from the phoenix read buffer.

INFSMART_Phoenix_EmptyFifo

Flushes the phoenix receivebuffer.

Prototype:
```
SDK_STATUS INFSMART_Phoenix_EmptyFifo(HANDLE hDevice);
```

Parameters:
1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_Trap

Traps a card, by sending the specified command to the card after resetting a card and delaying for the specified number of microseconds (us). This command should be used for time critical commands. Typically this command could be used with TitaniumCard where the delay parameter should be 0x32 (0x32*10 = 500us) and Trapvalue should be 0x55.

The sequence this commands carries out is as follows:

1. Set reset
2. Clear reset
3. Wait 'Delay'*10 us
4. Send 'Trapvalue' command through phoenix

Prototype:
SDK_STATUS INFSMART_Phoenix_Trap(HANDLE hDevice, unsigned char Delay, unsigned char Trapvalue);

Parameters:
1. hDevice: Handle of the device.
2. Delay: Specifies the delay in us*10 (Delay = 50, equals 500us), between resetting the card and sending the command.
3. Trapvalue: The command to send after the delay.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_TrapDelay

This function is identical to the INFSMART_Phoenix_Trap, but it delays for the specified number of milliseconds before returning from trapping.

Prototype:
SDK_STATUS INFSMART_Phoenix_TrapDelay(HANDLE hDevice, unsigned char Delay, unsigned char Trapvalue, DWORD dwSleep);
Parameters:
1. hDevice: Handle of the device.
2. Delay: Specifies the delay in us*10 (Delay = 50, equals 500us), between resetting the card and sending the command.
3. Trapvalue: The command to send after the delay.
4. dwSleep: Specifies the delay the function waits before it returns.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

INFSMART_Phoenix_BreakTrap

Traps a card, by sending the specified command to the card after resetting a card and delaying for the specified number of ms. This command should be used for time critical commands where the communications line should be in a break-state before the card is out of reset. Typically this command could be used with KnotCard where the delay parameter should be 0x28 (0x28*10 = 400ms) and Trapvalue should be 0x57.

The sequence this commands carries out is as follows:

1. Set reset
2. Set communications state to break-state.
3. Clear reset
4. Wait 'Delay'*10 MS
5. Send 'Trapvalue' command through phoenix

Prototype:
SDK_STATUS INFSMART_Phoenix_BreakTrap(HANDLE hDevice, unsigned char Delay, unsigned char Trapvalue);

Parameters:
1. hDevice: Handle of the device.
2. Delay: Specifies the delay in ms*10 (Delay = 50, equals 500ms), between resetting the card and sending the command.
3. Trapvalue: The command to send after the delay.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_PHOENIX_NOT_ENABLED (0x60)

Communications

INFSMART_Write

Writes the specified number of bytes from the specified buffer to the specified device. Given valid parameters, this function is blocking until the write is successful, fails, or a timeout occurs. The write is successful when the device has accepted all of the data. If the write fails or a timeout occurs, SDK_WRITE_ERROR is returned (see INFSMART_SetTimeouts).

Prototype:
```
SDK_STATUS INFSMART_Write(HANDLE hDevice, LPCVOID lpBuffer, DWORD
nNumberOfBytesToWrite, LPDWORD lpdwBytesWritten);
```

Parameters:
1. hDevice: Handle of the device.
2. lpBuffer: Address of a buffer of data to write.
3. nNumberOfBytesToWrite: Number of bytes to write to the device (0-4096 bytes)
4. lpdwBytesWritten: Address of a DWORD which will contain the number of bytes actually written to the device.

Return values:
```
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_WRITE_ERROR (0x04) or
SDK_INVALID_REQUEST_LENGTH (0x07) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06)
```

INFSMART_Read

Reads the specified number of bytes into the specified buffer and retrieves the number of bytes read. Given valid input parameters, this function is blocking until the specified number of bytes become available or a timeout occurs (see INFSMART_SetTimeouts).

Prototype:
```
SDK_STATUS INFSMART_Read(HANDLE hDevice, LPVOID lpBuffer, DWORD
nNumberOfBytesToRead, LPDWORD lpdwBytesRead);
```

Parameters:
1. hDevice: Handle of the device.
2. lpBuffer: Address of a buffer to receive the data.
3. nNumberOfBytesToRead: Number of bytes to read from the device (0-64Kbytes)
4. lpdwBytesRead: Address of a DWORD which will contain the number of bytes actually read from the device.

Return values:
```
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_READ_ERROR (0x02) or
SDK_INVALID_REQUEST_LENGTH (0x07) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06) or
SDK_RX_QUEUE_NOT_READY (0x03)
```

INFSMART_BytesInFifo

Returns the number of bytes ready to be read from the internal buffer.

Prototype:
DWORD INFSMART_BytesInFifo(HANDLE hDevice);

Parameters:
1. hDevice: Handle of the device.

Return values:
Number of bytes ready to be read.

INFSMART_EmptyFifo

Flushes the internal receivebuffer and the transmitbuffer of the device.

Prototype:
SDK_STATUS INFSMART_EmptyFifo(HANDLE hDevice);

Parameters:
1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_DEVICE_IO_FAILED (0x08)

Synchronous communications

SDK_STATUS INFSMART_SynchronousBegin

Powers on the card, and resets the card ready for reading the ATR.

```
Prototype:
SDK_STATUS INFSMART_SynchronousBegin(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousEnd

Powers down the card and ends synchronous communications.

```
Prototype:
SDK_STATUS INFSMART_SynchronousEnd(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousProcess

Clocks CLK until IO changes state to the value specified by iostate.

```
Prototype:
SDK_STATUS INFSMART_SynchronousProcess(HANDLE hDevice, unsigned char iostate);

Parameters:
   1. hDevice: Handle of the device.
   2. Iostate: The state of the IO line, when CLK should stop

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousWriteCommand

Writes command, address and data to the card.

```
Prototype:
SDK_STATUS INFSMART_SynchronousWriteCommand(HANDLE hDevice, unsigned char
control, unsigned char adr, unsigned char data);

Parameters:
   1. hDevice: Handle of the device.
   2. Control: The control byte
```

```
   3. Adr     : The address
   4. Data      : The data to send

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousWriteOneByteLSB

Synchronously clocks out one byte LSB first.

```
Prototype:
SDK_STATUS INFSMART_SynchronousWriteOneByteLSB(HANDLE hDevice, unsigned char
data);


Parameters:
   1. hDevice: Handle of the device.
   2. Data    : Data byte to clock out

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousWriteOneByteMSB

Synchronously clocks out one byte MSB first.

```
Prototype:
SDK_STATUS INFSMART_SynchronousWriteOneByteMSB(HANDLE hDevice, unsigned char
data);


Parameters:
   1. hDevice: Handle of the device.
   2. Data    : Data byte to clock out

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousCycleClock

Cycles CLK one time.

```
Prototype:
SDK_STATUS INFSMART_SynchronousCycleClock(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousRead8bit

Reads count amount of 8bits data.

```
Prototype:
SDK_STATUS INFSMART_SynchronousRead8bit(HANDLE hDevice,unsigned char count,
unsigned char* data);
Parameters:
   1. hDevice: Handle of the device.
   2. Count  : The number of bytes to read
   3. Data   : Pointer to buffer to store the data

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousReadXbytesYbits

Reads 'bytecount' amount of bytes with 'bitcount' amount of bits in each byte. 'Bitcount' should be less than or equal to 8bits. For reading 9 or 10 bits, combine **SynchronousRead8bit** and **SynchronousReadXbytesYbits.**

```
Prototype:
SDK_STATUS INFSMART_SynchronousReadXbytesYbits(HANDLE hDevice,unsigned char
bytecount, unsigned char bitcount, unsigned char* data);

Parameters:
   1. hDevice: Handle of the device.
   2. Bytecount : The amount of bytes to read
   3. Bitcount  : The amount of bits in each byte

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_SynchronousSetRST

Sets RST to the desired state.

```
Prototype:
SDK_STATUS INFSMART_SynchronousSetRST(HANDLE hDevice, unsigned char state);

Parameters:
   1. hDevice: Handle of the device.
   2. State  : The state of RST, low or high.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

I2C communications

SDK_STATUS INFSMART_IICClockOutStart

Clocks out I2C start condition.

```
Prototype:
SDK_STATUS INFSMART_IICClockOutStart(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_IICClockOutStop

Clocks out I2C stop condition.

```
Prototype:
SDK_STATUS INFSMART_IICClockOutStop(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_IICClockOutByte
Clocks out one byte.

```
Prototype:
SDK_STATUS INFSMART_IICClockOutByte(HANDLE hDevice, unsigned char data);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_IICClockInByteAck

Clock in one byte with acknowledge.

```
Prototype:
SDK_STATUS INFSMART_IICClockInByteAck(HANDLE hDevice, unsigned char *data);

Parameters:
   1. hDevice: Handle of the device.
   2. Data   : Pointer to buffer, to receive data.
```

```
Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_IICClockInByteNack

Clock in one byte without acknowledge.

```
Prototype:
SDK_STATUS INFSMART_IICClockInByteNack(HANDLE hDevice, unsigned char *data);

Parameters:
   1. hDevice: Handle of the device.
   2. Data   : Pointer to buffer, to receive data.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

SDK_STATUS INFSMART_IICCycleClock

Cycles CLK one time.

```
Prototype:
SDK_STATUS INFSMART_IICCycleClock(HANDLE hDevice);

Parameters:
   1. hDevice: Handle of the device.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_DEVICE_IO_FAILED (0x08)
```

Version history

Rev. 1.0	30.03.2009	First public release