SDK 2.50 Programmers Guide

for

INFINITY® USB UNLIMITED

(firmware v1.05)

Introduction

The Infinity® USB Unlimited SDK provides a simple programming interface for the Infinity USB Unlimited. Besides the basic features of controlling the RGB-LED and detecting if a card is inserted, the SDK makes it easy to access the ISO7816 (phoenix) interface of the Infinity USB Unlimited.

System overview

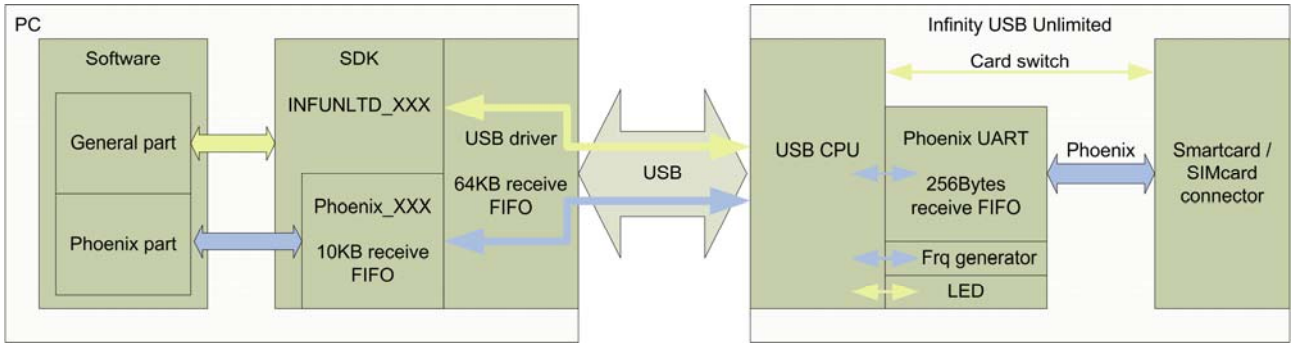

Figure 1. System overview

Typical software that uses the SDK contains a general part that handles finding (enumerate), opening and setup of the device and another part that handles the phoenix operations.

The general part should use the functions named INFUNLTD_XXX to find, open and setup the device. The phoenix part should then use the INFUNLTD_Phoenix_XXX functions to setup the frequency generator and the UART of the device and communicate with the card through the phoenix interface. The INFUNLTD_Phoenix_XXX functions uses the INFUNLTD_XXX functions to communicate with the device over USB.

When all phoenix communication is done, the phoenix interface and frequency generator should be closed down, and the device itself should be closed.

The phoenix UART consists of a 256 bytes receive buffer to handle incoming data from the card. The phoenix part of the SDK needs to empty this buffer before overflows occur and this is done by using the INFUNLTD_Phoenix_Read or INFUNLTD_Phoenix_BytesInFifo. The BytesInFifo function, copies data from the device to the internal 10KB receive FIFO and then returns the total amount of data available. If 256 bytes FIFO is not enough you should call INFUNLTD_Phoenix_Read continuously to empty the device fifo into its internal 10KB FIFO.

Communication

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

Data I/O through phoenix mode is performed using INFUNLTD_Phoenix_Write and INFUNLTD_Phoenix_Read functions. When phoenix mode operation is complete, phoenix mode is disabled by a call to INFUNLTD_Phoenix_Disable. Remember to disable the frequency generator by calling INFUNLTD_Phoenix_SetFrequency with a 0 as frequency.
When all I/O operations are complete, the device is closed by a call to INFUNLTD_Close.

Supporting multiple devices

This SDK supports multiple Infinity USB Unlimited 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 INFUNLTD_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.

There are two ways to support multiple devices.

1. One instance of your application can handle multiple devices, this is the most advanced and usually not needed.
2. One instance of your application can handle one device, but multiple instances of your application can be opened and handle each its own device. To support this you should start by finding the first device that is not currently in use.

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 cards are inserted.

- **Phoenix**
  The phoenix functions are used to control, read and write the ISO7816 interface and control the onboard frequency generator. 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 Unlimited.

- **Communications**
  These functions are used to read and write data directly to the Infinity USB Unlimited, 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

INFUNLTD_GetSDKVersion

Returns the version of the used SDK.

Prototype:
```
DWORD INFUNLTD_GetSDKVersion();
```

Parameters:
-None

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

INFUNLTD_GetNumDevices

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

Prototype:
```
SDK_STATUS INFUNLTD_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)

INFUNLTD_OpenDeviceFromNum

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

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

Parameters:
1. dwDevice: Zero-based index of the device to open. Use INFUNLTD_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)

INFUNLTD_CloseDevice

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

Prototype:
SDK_STATUS INFUNLTD_CloseDevice(HANDLE hDevice);

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

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

INFUNLTD_SetTimeouts

Sets the read and write timeouts.

The timeouts set are valid only for subsequent calls to INFUNLTD_Write, INFUNLTD_Phoenix_Write and INFUNLTD_Read from the same thread that called INFUNLTD_SetTimeouts. This is only an issue if you have a multithreaded application.

Prototype:
SDK_STATUS INFUNLTD_SetTimeouts(DWORD dwRead, DWORD dwWrite);

Parameters:
1. dwRead: Read timeout. Timeout in milliseconds for read operations.
2. dwWrite: Write timeout. Timeout in milliseconds for write operations.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00)

INFUNLTD_SetLEDState

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

Prototype:
SDK_STATUS INFUNLTD_SetLEDState(HANDLE hDevice, unsigned int R, unsigned int G,unsigned int B, 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: 16bit value specifying the amount of BLUE. (0x0000 = Off, 0xFFFF = On)
5. PWMFrq: 8bit value specifying the PWM frequency of the LED. (0x00 = Slow, 0xFF = Fast)

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

INFUNLTD_GetCardState

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

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_GetFirmwareVersion

Returns the version of the firmware in the programmer.

Prototype:
DWORD INFUNLTD_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.

INFUNLTD_SetVoltage

Sets the programming voltage to either 3.3v or 5.5v.

Prototype:
DWORD INFUNLTD_SetVoltage(HANDLE hDevice, dwVolt);
Parameters:
1. hDevice: Handle of the device.
2. A DWORD specifying the desired voltage. Use 5 for 5v programming voltage or 3 for 3.3v programming voltage. The programmer defaults to 5v when connected to the USB port.

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

PHOENIX

INFUNLTD_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 INFUNLTD_Phoenix functions.

Prototype:
SDK_STATUS INFUNLTD_Phoenix_Enable(HANDLE hDevice, unsigned int baud, unsigned int stopbitparity);

Parameters:

1. hDevice: Handle of the device.
2. baud: Value indicating which baudrate the UART uses.
   The predefined values are:
   0 – PHOENIX_BAUD_2400
   1 – PHOENIX_BAUD_9600
   2 – PHOENIX_BAUD_19200
   3 - PHOENIX_BAUD_28800
   4 – PHOENIX_BAUD_38400
   5 – PHOENIX_BAUD_57600
   6 – PHOENIX_BAUD_115200
3. stopbitparity: Number Value indicating which stopbit and parity the UART uses.

   The low order of the byte is the parity

   The predefined values are:
   0x00 – PHOENIX_PARITY_NONE
   0x01 – PHOENIX_PARITY_ODD
   0x02 – PHOENIX_PARITY_EVEN
   0x03 - PHOENIX_PARITY_MARK
   0x04 - PHOENIX_PARITY_SPACE

   The high order of the byte is the stop bit

   The predefined values are:
   0x00 - PHOENIX_ONESTOPBIT (1 stopbit)
   0x20 - PHOENIX_TWOSTOPBIT (2 stopbits)

   When passing the stopbit and parity the bitwise OR operator can be used. For instance (PHOENIX_TWOSTOPBIT | PHOENIX_PARITY_EVEN)

ISO7816 normally uses 9600bps with EVEN parity.

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

INFUNLTD_Phoenix_Change

Changes baud-rate and parity properties for a device which already has been phoenix enabled with INFUNLTD_Phoenix_Enable. This function should not be called prior to INFUNLTD_Phoenix_Enable.

Prototype:
SDK_STATUS INFUNLTD_Phoenix_Change(HANDLE hDevice, unsigned int baud, unsigned int stopbitparity);

Parameters:

1. hDevice: Handle of the device.
2. baud: Value indicating which baudrate the UART uses.
   The predefined values are:
   0 – PHOENIX_BAUD_2400
   1 – PHOENIX_BAUD_9600
   2 – PHOENIX_BAUD_19200
   3 - PHOENIX_BAUD_28800
   4 – PHOENIX_BAUD_38400
   5 – PHOENIX_BAUD_57600
   6 – PHOENIX_BAUD_115200
3. stopbitparity: Number Value indicating which stopbit and parity the UART uses.

   The low order of the byte is the parity

   The predefined values are:
   0x00 – PHOENIX_PARITY_NONE
   0x01 – PHOENIX_PARITY_ODD
   0x02 – PHOENIX_PARITY_EVEN
   0x03 - PHOENIX_PARITY_MARK
   0x04 - PHOENIX_PARITY_SPACE

   The high order of the byte is the stop bit

   The predefined values are:
   0x00 - PHOENIX_ONESTOPBIT (1 stopbit)
   0x20 - PHOENIX_TWOSTOPBIT (2 stopbits)

   When passing the stopbit and parity the bitwise OR operator can be used. For instance (PHOENIX_TWOSTOPBIT | PHOENIX_PARITY_EVEN)

ISO7816 normally uses 9600bps with EVEN parity.

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

INFUNLTD_Phoenix_ChangeCustomBaud

Changes baud-rate and parity properties for a device which already has been phoenix enabled with INFUNLTD_Phoenix_Enable. This function should not be called prior to INFUNLTD_Phoenix_Enable. This function is similar to INFUNLTD_Phoenix_Change, but adds the possibility of using non-standard baudrates.

Prototype:
SDK_STATUS INFUNLTD_Phoenix_ChangeCustomBaud(HANDLE hDevice, unsigned int baud, unsigned int * ActualBaud, unsigned int stopbitparity);
Parameters:

1. hDevice: Handle of the device.

2. baud: Value indicating which baudrate the UART should use within the range of 1200 – 230400 baud.

3. ActualBaud: Address of an unsigned int which will contain the nearest match of the desired baudrate, that the programmer can generate.

4. stopbitparity: Number Value indicating which stopbit and parity the UART uses.

   The low order of the byte is the parity

   The predefined values are:
   0x00 – PHOENIX_PARITY_NONE
   0x01 – PHOENIX_PARITY_ODD
   0x02 – PHOENIX_PARITY_EVEN
   0x03 - PHOENIX_PARITY_MARK
   0x04 - PHOENIX_PARITY_SPACE

   The high order of the byte is the stop bit

   The predefined values are:
   0x00 - PHOENIX_ONESTOPBIT (1 stopbit)
   0x20 - PHOENIX_TWOSTOPBIT (2 stopbits)

   When passing the stopbit and parity the bitwise OR operator can be used. For instance (PHOENIX_TWOSTOPBIT | PHOENIX_PARITY_EVEN)

ISO7816 normally uses 9600bps with EVEN parity.

Return values:
SDK_STATUS =
SDK_SUCCESS (0x00) or
SDK_INVALID_HANDLE (0x01) or
SDK_INVALID_PARAMETER (0x06) or
K_PHOENIX_NOT_ENABLED (0x60)

INFUNLTD_Phoenix_Disable

Disables phoenix mode and returns to normal state, remember to set the frequency generator to 0 using INFUNLTD_Phoenix_SetFrequency.

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_Phoenix_Write

This function uses INFUNLTD_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 INFUNLTD_Phoenix_Enable function. Phoenix mode has to be enabled using INFUNLTD_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.

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 (see INFUNLTD_SetTimeouts).

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_Phoenix_Read

This function uses INFUNLTD_Read and INFUNLTD_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 INFUNLTD_Phoenix_Enable function.
Phoenix mode has to be enabled using INFUNLTD_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 INFUNLTD_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 INFUNLTD_Phoenix_BytesInFifo, prior to calling this function to make sure the requested bytes are actually ready to be read.

Prototype:
```
SDK_STATUS INFUNLTD_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)

INFUNLTD_Phoenix_SetFrequency

Enables the onboard frequency generator and sets the output to the specified frequency or the closest match. This function calculates properties of the frequency generator to match the desired frequency as closely as possible. Not all odd frequencies can be matched, for the most used frequencies use the predefined values for the parameter dwFrq. You should set the frequency to 0 to disable the frequency generator.

Prototype:
SDK_STATUS INFUNLTD_Phoenix_SetFrequency(HANDLE hDevice, DWORD dwFrq);

Parameters:
1. hDevice: Handle of the device.
2. dwFrq: Value indicating which frequency to set the frequency generator to in Hz. You can specify any frequency from 1 MHz to 25 MHz, but not all frequencies are guarantied to be as precise as the predefined values. The predefined values are the most used frequencies.

   The predefined values are:
   3579000 = PHOENIX_FRQ_3579000
   3680000 = PHOENIX_FRQ_3680000
   6000000 = PHOENIX_FRQ_6000000

ISO7816 normally uses a frequency of 3579000Hz (3.58Mhz).

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

INFUNLTD_Phoenix_SetRST

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

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_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 INFUNLTD_Phoenix_Read afterwards to read the ATR that usually follows a card-reset from the card. Use the INFUNLTD_Phoenix_BytesInFifo function to determine the size of the ATR to read.

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_Phoenix_BytesInFifo

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

Prototype:
DWORD INFUNLTD_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.

INFUNLTD_Phoenix_EmptyFifo

Flushes the phoenix receivebuffer.

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_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 INFUNLTD_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)

INFUNLTD_Phoenix_TrapDelay

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

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_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 INFUNLTD_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

INFUNLTD_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 INFUNLTD_SetTimeouts).

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_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 INFUNLTD_SetTimeouts).

Prototype:
SDK_STATUS INFUNLTD_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)

INFUNLTD_BytesInFifo

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

Prototype:
DWORD INFUNLTD_BytesInFifo(HANDLE hDevice);

Parameters:
1. hDevice: Handle of the device.

Return values:
Number of bytes ready to be read.

INFUNLTD_EmptyFifo

Flushes the internal receivebuffer and the transmitbuffer of the device.

Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousBegin
Powers on the card, and resets the card ready for reading the ATR.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousEnd
Powers down the card and ends synchronous communications.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousProcess
Clocks CLK until IO changes state to the value specified by iostate.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousWriteCommand
Writes command, address and data to the card.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousWriteOneByteLSB
Synchronously clocks out one byte LSB first.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousWriteOneByteMSB
Synchronously clocks out one byte MSB first.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousCycleClock
Cycles CLK one time.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousRead8bit
Reads count amount of 8bits data.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousReadXbytesYbits
Reads 'bytecount' amount of bytes with 'bitcount' amount of bits in each byte. 'Bitcount' should be less than or equal to 16bits. Returns 2 bytes for each bytecount.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_SynchronousSetRST
Sets RST to the desired state.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICClockOutStart
Clocks out I2C start condition.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICClockOutStop
Clocks out I2C stop condition.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICClockOutByte
Clocks out one byte.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICClockInByteAck
Clock in one byte with acknowledge.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICClockInByteNack
Clock in one byte without acknowledge.

```
Prototype:
SDK_STATUS INFUNLTD_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 INFUNLTD_IICCycleClock
Cycles CLK one time.

```
Prototype:
SDK_STATUS INFUNLTD_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. 1.32	17.03.2008	Changes for SDK2.50: Updated for use with latest USB drivers. Added support for SLE and I2C memory cards.
Rev. 1.31	28.05.2005	Changes for SDK2.07: Fixed bug: Stopbits not being set correct (always 2 stopbits).
Rev. 1.3	19.05.2005	Changes for SDK2.06: Added the option of specifying stopbits in phoenix mode. Added INFUNLTD_GetFirmwareVersion which returns the firmware version of the programmer. Added INFUNLTD_SetVoltage which sets the programming voltage of the programmer 3.3v or 5v. Added INFUNLTD_Phoenix_TrapDelay, which adds more options for trapping cards. Added INFUNLTD_Phoenix_ChangeCustomBaud, which makes it possible to use non-standard baudrates for the phoenix interface. *SDK2.06 requires firmware v1.04 or higher.*
Rev. 1.2	16.03.2005	Minor changes: Changed INFUNLTD_Phoenix_SetRST, bool parameter changed to unsigned int. Added error code SDK_PHOENIX_NOT_ENABLED (0x60) to Phoenix functions. Changed parity values to match windows defined parity values.
Rev. 1.1	11.03.2005	First public release