unitech

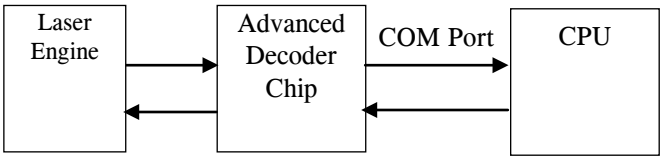
HT660e/HT680/PA968 Series/PA550/PA600 Programming Manual

V 1.21 2011-04-18

1 Introduction

1.1 How to download data from scanner

The major difference between the HT6xx/PA96x/RH_Series/PA982 and a standard HPC/PalmPC is barcode input capability. The WinCE Reference Manual contains no information regarding barcode input. This section will introduce the programming structure of the barcode sub-system and the programming utility library for the device. Inside the device there is an advanced decoding chip to control SE900 laser engine and to handle barcode decoding. Below is system diagram for the device's barcode:



According to the above diagram, the device communicates with Decoder Chip by mean of serial port. Its communication parameter is fixed on 38400,N,8.1. Normally, the Decoder Chip is in sleep mode when COM port is not activated. When COM port is activated, the Decoder Chip will start working, and it will decode the barcode "signal" from the laser engine when the trigger key is pressed. After decoding, barcode data and its symbology type will be sent directly to the device.

Many programmers find it difficult to control the Decoder Chip via programming language alone, especially if they are not familiar with barcode and serial port controls. Because of this, Unitech provides the following utility library and program for the user or application programmer to control the Decoder Chip:

1. Application program "Scan2Key.exe" is a useful application program that can read input data from the laser scanner and then directly input the data into device's keyboard buffer. "Scan2Key.exe" makes barcode data input simple, and can be especially valuable to those programmers not familiar with COM port programming. User program simply reads the barcode data from the keyboard. For barcode symbologies setting, you can run **Scanner Setting** from **Control Panel** to define all of supporting symbologies and delimiter.

2. Utility library:
   For programming control, HT6xx/PA96x/PA982 provides USI.DLL to let user control scanner input, symbologies setting and profile controlling. Please refer to 2 for detail API lists.

   USI.DLL is Unitech's new scanner function library on the device. For backward compatible issue, Unitech still provide Scanner3.DLL and ScanKey3.DLL for existing PT930/PT930SA user to port their software into the device, but several APIs on Scanner3.DLL and ScanKey3.DLL have already been removed on the other devices.

1.2 COM definition for HT6xx/PA96x/PA982/RH Series/PA6xx/PA550

General*:*

COM 1	Physical full RS232 port (ActiveSync)
COM 2	Scanner (Hamster) or RFID reader
COM 3	IrComm
COM 4	USB client
COM 5	IrDA or Bluetooth
COM 6	Reserve
COM 7	Bluetooth Printer
COM 8	Bluetooth Modem
COM 9	Bluetooth ActiveSync

PA968*:*

COM 0	USB modem
COM 1	Bluetooth
COM 2	Scanner (Hamster) or RFID reader
COM 3	Reserve
COM 4	Reserve
COM 5	GPS
COM 6	Reserve
COM 7	Bluetooth Printer
COM 8	Bluetooth Modem
COM 9	Bluetooth ActiveSync

HT680:

COM 1	Bluetooth
COM 2	Scanner (Hamster)
COM 3	Reserve
COM 4	USB Client
COM 5	Reserve
COM 6	GPRS
COM 7	Bluetooth Printer
COM 8	Bluetooth Modem
COM 9	Bluetooth ActiveSync

PA600 :

Mobile Version:

COM 0	USB to serial
COM 1	Reserve
COM 2	Bluetooth
COM 3	IrDAComm
COM 4	Scanner (Hamster)
COM 5	BTModem
COM 6	USB client
COM 7	Reserve
COM 8	Reserve
COM 9	RawIR / RFID

CE Version:

COM 0	USB to serial
COM 1	Bluetooth
COM 2	RFID/Scanner (Hamster)
COM 3	IrDAComm
COM 4	USBClient
COM 5	RawIR
COM 6	Reserve
COM 7	BT Modem
COM 8	BT Printer
COM 9	BT ActiveSync

PA968II/PA690 :

COM 0	Scanner (Hamster)
COM 1	USB to serial
COM 2	Bluetooth
COM 3	Reserve
COM 4	GPS(Virtual Port)
COM 5	BT Modem
COM 6	USB client
COM 7	Bluetooth BTHATCI server
COM 8	Reserve
COM 9	Reserve

PA550 :

COM 0	Scanner (Hamster)
COM 1	USB to serial
COM 2	Bluetooth
COM 3	GPS
COM 4	GPS(Virtual Port)
COM 5	BT Modem
COM 6	USB Client
COM 7	Bluetooth BTHATCI server
COM 8	Reserve
COM 9	Reserve

1.3 Get SDK from Unitech?

You can get WinCE SDK from below URL

HT660	SDK http://w3.tw.ute.com/pub/cs/sdk/ht660/HT660SDK.zip
HT680	SDK http://w3.tw.ute.com/pub/cs/sdk/HT680/HT680SDK.zip
HT660II/660e	SDK http://w3.tw.ute.com/pub/cs/SDK/HT660II/HT660eSDK.zip
HT660IICore	SDK http://w3.tw.ute.com/pub/cs/SDK/HT660II/HT660IICore.zip
PA962/963	SDK http://w3.tw.ute.com/pub/cs/sdk/pa962/pa962sdk.zip
PA966/967	SDK http://w3.tw.ute.com/pub/cs/sdk/pa966/pa966sdk.zip
PA968	SDK http://w3.tw.ute.com/pub/cs/sdk/pa968/pa968sdk.zip
PA982	SDK http://w3.tw.ute.com/pub/cs/sdk/pa982/Pa982SDK.zip
RH767	SDK http://w3.tw.ute.com/pub/cs/sdk/RH767/RH767_CE5_SDK.zip
PA600	SDK http://w3.tw.ute.com/pub/cs/SDK/PA600/PA600_CE5_SDK.zip

2 USI.DLL – Unitech Scanner Interface DLL

The link includes the sample program and SDK for USI.
http://w3.tw.ute.com/pub/cs/SDK/USI/USISDK.zip

Note : For programming, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide *.H and *.LIB for compiler for Windows Mobile OS), please refer to Chapter 5 for programming guide.

2.1 Register the application to the USI DLL

Function Description:

Register the application to the USI DLL, so that the DLL can communicate with the application. It will also open and initial scanner port (COM2, for example) and set the scanner to the working mode. The application should call USI_Unregister to unregister from the DLL after done with the scanner.

Function call:

```
BOOL USI_Register(HWND hwnd, UINT msgID);
```

Parameter: (input)

hwnd: Handle of the window to which USI DLL will send messages to report all activities, including error messages, scan data ready, etc.

msgID: Specifies the message to be posted. DLL will post messages by calling: PostMessage(hwnd, msgID, msg, param).

The window procedure will receive custom message about msgID and wParam parameter can be one of the followings:

SM_ERROR_SYS:	Indicates a system error, which is caused by a call to the system function. Param contains the error code from GetLastEror().
SM_ERROR	Indicates an error. Param contains the cause of error, which can be on of followings:

SERR_INVALID_HWND:	Invalid window handle.
SERR_INVALID_MSGID:	msgID cannot be 0.
SERR_OPEN_SCANNER:	Open or initial scanner port failed.
SERR_CHECKSUM:	Checksum error in received packet.
SERR_DATALOST:	New scan data is lost because data buffer is not empty.
SERR_BUFFEROVERFLOW:	Data buffer overflow. The default size is 4K bytes.

SM_REPLY	Indicates received a reply. All the responses from the scanner except the scan data will be notified by this message.
SM_DATAREADY	Indicates that scan data is successfully decoded and ready to retrieve.
SM_ACK	Indicates received a ACK.
SM_NAK	Indicates received a NAK.
SM_NOREAD	Indicates received a No-Read packet.

Note: Scanner port settings are defined in registry as described below:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Unitech America Inc.\Scanner\Settings]
"COMPORT"="COM2:"
"BAUDRATE"="38400"
"STOPBITS"="1"
"PARITY"="None"
"CHECKPARITY"="1"
```

Return:

BOOL: TRUE : OK
FALSE : Failure

2.2 Unregister the application from the USI.DLL

Function Description:

Unregister the application from the DLL. It will close the scanner port, and by default it will disable the scanner.

Function call:

```
void USI_Unregister();
```

Return code:

None

2.3 Enable / Disable Scanner

Function Description:

To start or stop USI function. This function is useful for application to temporarily stop scanner function if it is only need keypad input or keep clear input buffer.

Function call:

```
BOOL USI_EnableScan(BOOL bStatus);
```

Parameter: (input)

bStatus: TRUE : Enable Scanner
FALSE : Disable Scanner

Return:

BOOL : TRUE : OK
FALSE : Failure

2.4 Reset Scanner

Function Description:

Set the scanner to the working mode, and reset the communication control.

Function call:

```
BOOL USI_Reset();
```

Return:

Always TRUE

2.5 Get error code

Function Description:

Returns the error code (SERR_***).

Function call:

```
DWORD USI_GetError();
```

Return:

Returns the error code (SERR_***), which has been described in USI_Register function.

2.6 *Returns the system error code*

Function Description:

Returns the system error code, which is returned by GetLastError. It will also return the description of the error in buffer if it is not NULL.

Function call:

```
DWORD USI_GetLastSysError(LPTSTR buffer, int len);
```

Parameter: (output)

buffer:	LPTSTR:	Data buffer for error message
len:	int:	Length of error message.

Return:

Returns the system error code, which is returned by system function GetLastError. It will also return the description of the error in buffer retrieved by system function FormatMessage if it is not NULL.

For a complete list of error codes, refer to the SDK header file WINERROR.H.

2.7 *Get scan data*

Function Description:

Retrieves the scan data into the buffer. Returns the length of characters. It also returns the barcode type if type is not NULL. Return 0 means that the buffer is too short to hold the data.
USI_GetData should be called when SM_DATAREADY message is received. Or call USI_ResetData to discard the data. Both of them will reset the data buffer so that next scan data can come in.
If the data buffer is not empty and a new scan data occurs, it will be discarded and an error message SM_ERROR with code of SERR_DATALOST will be sent.

Function call:

```
UINT USI_GetData(LPBYTE buffer, UINT len, UINT* type);
```

Parameter: (input)

len :	UINT:	Len specifies the maximum length of the buffer.

Parameter: (output)

buffer:	LPBYTE:	Data buffer for storing scanned data
type:	UINT:	barcode type which is defined on USI.H.

Please refer to below list

```
BCT_CODE_39                 // Code 39
BCT_CODABAR                 // CodaBar
BCT_CODE_128                // Code 128
BCT_INTERLEAVED_2OF5        // Interleaves 2 of 5
BCT_CODE_93                 // Code 93
BCT_UPC_A                   // UPC A
BCT_UPC_A_2SUPPS            // UPC A with 2 Supps
BCT_UPC_A_5SUPPS            // UPC A with 5 Supps
BCT_UPC_E0                  // UPC E
BCT_UPC_E0_2SUPPS           // UPC E with 2 Supps
BCT_UPC_E0_5SUPPS           // UPC E with 5 Supps
BCT_EAN_8                   // EAN 8
BCT_EAN_8_2SUPPS            // EAN 8 with 2 Supps
BCT_EAN_8_5SUPPS            // EAN 8 with 5 Supps
BCT_EAN_13                  // EAN 13
BCT_EAN_13_2SUPPS           // EAN 13 with 2 Supps
BCT_EAN_13_5SUPPS           // EAN 13 with 5 Supps
BCT_MSI_PLESSEY             // MSI Plessey
```

```
BCT_EAN_128              // EAN 128
BCT_UPC_E1               // UPC E1
BCT_UPC_E1_2SUPPS        // UPC E1 with 2 Supps
BCT_UPC_E1_5SUPPS        // UPC E1 with 5 Supps
BCT_TRIOPTIC_CODE_39     // TRIOPTIC CODE 39
BCT_BOOKLAND_EAN         // Bookland EAN
BCT_COUPON_CODE          // Coupon Code
BCT_STANDARD_2OF5        // Standard 2 of 5
BCT_CODE_11_TELPEN       // Code 11 Telpen
BCT_CODE_32              // Code 32
BCT_DELTA_CODE           // Delta Code
BCT_LABEL_CODE           // Label Code IV & V
BCT_PLESSEY_CODE         // Plessey Code
BCT_TOSHIBA_CODE         // Toshiba Code China Postal Code / Matrix 2 of 5
```

Return:

UINT: Data length

2.8 *Get length of scanned data*

Function Description:

Returns the data length of the scan data. When allocate the memory to hold the scan data, add at least one additional byte for string terminator.

Function call:

```
UINT USI_GetDataLength();
```

Return:

UNIT: data length

2.9 *Get Symbology name*

Function Description:

Returns the barcode name of the type.

Function call:

```
LPCTSTR USI_GetBarcodeName(UINT type, LPBYTE buffer, UINT len);
```

Parameter: (input)

type: UINT : barcode type. (refer to 0 for type definition

buffer: LPBYTE : Please refer to below table

Type	Buffer
BCT_CODE_39	Code 39
BCT_CODABAR	Codabar
BCT_CODE_128	Code 128
BCT_INTERLEAVED_2OF5	Interleaved 2 of 5
BCT_CODE_93	Code 93
BCT_UPC_A	UPC A
BCT_UPC_A_2SUPPS	UPC A with 2 Supps.
BCT_UPC_A_5SUPPS	UPC A with 5 Supps.
BCT_UPC_E0	UPC E
BCT_UPC_E0_2SUPPS	UPC E with 2 Supps.
BCT_UPC_E0_5SUPPS	UPC E with 5 Supps.
BCT_EAN_8	EAN 8
BCT_EAN_8_2SUPPS	EAN 8 with 2 Supps.
BCT_EAN_8_5SUPPS	EAN 8 with 5 Supps.
BCT_EAN_13	EAN 13
BCT_EAN_13_2SUPPS	EAN 13 with 2 Supps.

BCT_EAN_13_5SUPPS	EAN 13 with 5 Supps.
BCT_MSI_PLESSEY	MSI Plessey
BCT_EAN_128	EAN 128
BCT_TRIOPTIC_CODE_39	Trioptic Code 39
BCT_BOOKLAND_EAN	Bookland EAN
BCT_COUPON_CODE	Coupon Code
BCT_STANDARD_2OF5	Standard 2 of 5
BCT_CODE_11_TELPEN	Code 11 or Telpen
BCT_CODE_32	Code 32 (Pharmacy Code)
BCT_DELTA_CODE	Delta Code
BCT_LABEL_CODE	Label Code IV & V
BCT_PLESSEY_CODE	Plessey Code
BCT_TOSHIBA_CODE	Toshiba Code (China Postal Code), Matrix 25

len: UINT : length of string on the 2nd parameter buffer

Return:

TRUE: if it found name for the barcode type,
FALSE: if not (type may be wrong

2.10 Clear scan data system buffer

Function Description:

Reset the data buffer so that next new scan data can come in.

Function call:

```
void USI_ResetData();
```

2.11 Good read indicator

Function Description:

Inform a good receiving of scan data, this will play a sound (wave file scanok.wav) and light the LED lasting for 1 second.

Function call:

```
void USI_ReadOK();
```

Note:

USI will call the function GoodReadLEDOn function exported by the DLL defined in the registry described below (UPI300.DLL is an example) to turn on and off the LED. If the DLL is not defined or the function is not found, USI will bypass the call of GoodReadLEDOn.

[HKEY_LOCAL_MACHINE\SOFTWARE\Unitech America Inc.\Scanner\Settings]
"DLLLEDCONTROL"="UPI300.DLL"

The function prototype of GoodReadLEDOn is:
VOID WINAPI GoodReadLEDOn(BOOL fon);
Turn on when fon is TRUE, and turn off when fon is FALSE.

2.12 Wait for acknowledgement of the last sent command

Function Description:

Wait for acknowledgement of the last sent command until timeout. It is useful when a serial of commands needs to be sent at a time. Before call USI_SendCommand, call USI_WaitForSendEchoTO to make sure that the previous command is done.

Function call:

```
BOOL USI_WaitForSendEchoTO(DWORD timeout);
```

Parameter: (input)

timeout: DWORD : Specifies the timeout in millisecond.

Return:

Returns FALSE if timeout.

2.13 Save setting to profiles

Function Description:

Save current settings of scanner so that the settings will be persistent when the unit get power off and on again.

Function call:

```
BOOL USI_SaveCurrentSettings();
```

Return :

TRUE: Success,
FALSE: Fail

2.14 Save scanner setting into specified file

Function Description:

Save the current settings to file. The file takes "*.USI" as extension name.

Function call:

BOOL USI_SaveSettingsToFile(LPCTSTR filename)

Parameter: (input)

filename : LPCTSTR: file name for setting profile

Return:

TRUE: Success,
FALSE: Fail

2.15 Change scanner setting from specified setting profile

Function Description:

Load and activate the settings from file.

Function call:

BOOL USI_LoadSettingsFromFile(LPCTSTR filename, BOOL formulaOnly);

Parameter: (input)

filename: LPCTSTR : name of scanner setting profile (*.USI)
formulaOnly: BOOL: if TRUE, only data editing formulas are load. The other settings remain unchanged

Return:

TRUE: Success,
FALSE: Fail

2.16 Automatically enable scanner beam with pressing trigger key

Function Description:

Start auto scanning. Scan engine will be automatically triggerrd on.

Function call:

BOOL USI_StartAutoScan(DWORD interval);

Parameter: (input)

interval: DWORD: Specifies the interval in milli-second

Return:

TRUE: Success,
FALSE: Fail

Note:

USI will call the function SetScannerOn function exported by the DLL defined in the registry described below (UPI300.DLL is an example) to start and stop the scanner. If the DLL is not defined or the function is not found, then auto scanning is not available.

[HKEY_LOCAL_MACHINE\SOFTWARE\Unitech America Inc.\Scanner\Settings]
"DLLSCANNERCONTROL"="UPI300.DLL"

The function prototype of SetScannerOn is:
VOID WINAPI SetScannerOn(BOOL fon);
Start when fon is TRUE, and stop when fon is FALSE.

2.17 Stop auto scanning function

Function Description:

Stop auto scanning

Function call:

```
void USI_StopAutoScan();
```

2.18 Check if auto scanning is enable

Function Description:

Check if auto scanning function is enabled or not

Function call:

```
BOOL USI_IsAutoScanning()
```

Return:

BOOL:	TRUE:	auto-scanning is running
	FALSE:	auto-scanning is disabled.

2.19 Check if Scan2Key.exe program is running or not

Function Description:

Test whether Scan2Key application is running at background. (It doesn't mean Scan2Key is routing scanner input to keyboard, please call S2K_IsEnabled() to check if routing function is enable or not)

Function call:

```
HWND S2K_IsLoaded();
```

Return:

NULL:	Scan2Key is not running
Non-NULL:	Indicates scan2key is running. It actually returns window handle for scan2key, but it is for internal use – send messages.

2.20 Test if Scan2Key is enabled

Function Description:

Test whether Scan2Key is enabled. Scan2Key routes scanning input from scanner to keypad buffer, so that barcode data can be input as like from keystrokes on keypad.

Function call:

```
BOOL S2K_IsEnabled();
```

Return:

TRUE:	Enabled.
FALSE:	Disabled

2.21 Load/Unload Scan2Key.exe

Function Description:

Load or unload Scan2Key

Function call:

```
BOOL S2K_Load(BOOL load, DWORD timeout);
```

Parameter: (input)

load:	BOOL:	TRUE = load Scan2Key FALSE = unload Scan2Key
timeout:	DWORD:	when unload Scan2Key, it will wait until Scan2Key has been removed from memory or timeout specified by this parameter.

Return:

TRUE = successfully loaded.

2.22 Enable/Disable Scan2Key

Function Description:

Enable or disable Scan2Key to put scanned data to standard keyboard input buffer. Scan2Key is enabled by default.

Function call:

```
BOOL S2K_Enable(BOOL enable, DWORD timeout);
```

Parameter: (input)

enable:	BOOL:	TRUE = Enable scanned data to keyboard buffer FALSE = Disable scanned data to keyboard
timeout:	DWORD:	when enable or disable Scan2Key, it will wait until Scan2Key has been removed from memory or timeout specified by this parameter.

Return:

TRUE:	if successfully enabled Scan2Key, otherwise FALSE

2.23 Send scanner command to decoding chip

Function Description:

Send scanner command to decoder chip. This command will send a serial of bytes to decoder chip as following: (Esc and BCC will be calculated and added automatically)

Esc, high-length, low-length, command-ID, operation, set, BCC

Please refer to complete command reference on section 4

```
BOOL HAM_SendCommand(BYTE highlen, BYTE lowlen, BYTE cmdID, BYTE op, BYTE set);
```

Parameter: (input)

highlen:	BYTE:	high byte of command length
lowlen:	BYTE:	low byte of command length
cmdID:	BYTE:	command ID
op:	BYTE:	operation mode for this command
set:	BYTE:	operand for this command

Return:

TRUE = Indicates the command has been successfully sent to queue to output.

2.24 *Only send single command decoding chip*

Function Description:

Send command to decoder chip. This is a variation of command HAM_SendCommand. It sends following command to Hamster: (note, only two bytes without BCC)

Esc, 0x80+cmd

Function call:

BOOL HAM_SendCommand1(BYTE cmd);

Parameter: (input)

cmd:	BYTE:	command

Return:

TRUE = indicates the command has been successfully sent to queue to output.

2.25 *Send command to decoding chip*

Function Description:

Send command to decoder chip. This is a variation of command HAM_SendCommand. It will read a number of parameters and packet them as in following format and send it to decoder chip.

Esc, parameter1, parameter2, …, BCC

The total number of parameters is specified by first parameter num.

Function call:

BOOL HAM_SendCommand2(BYTE num, BYTE parameter1, …);

Parameter: (input)

num:	BYTE:	number of total parameters
parameter*x*	BYTE:	Parameter

Return:

TRUE = indicates the command has been successfully sent to queue to output.

2.26 *Send scanner command set to decoding chip*

Function Description:

This function call has the same function as HAM_SendCommand except that it takes a single WORD parameter for the length and an extra timeout parameter. This is a synchronized function, it returns when command has been sent to and got response from scanner.
USI_WaitForSendEchoTO is not needed before a next continuous send command call.

To send a string please call HAM_SendCommand_SetString.

BOOL HAM_SendCommand_Set(WORD len, BYTE cmdID, BYTE op, BYTE set, DWORD timeout)

Parameter: (input)

len:	BYTE	: Specifies length of command, which is actually always be 4.
cmdID:	BYTE	: command ID
op:	BYTE	: operation mode for this command
set:	BYTE	: operand for this command
timeout:	DWORD	: Specifies the timeout in millisecond

Return:

TRUE = indicates the setting has been set successfully.

2.27 *Get scanner command set from decoding chip*

Function Description:

This function call has the similar function as HAM_SendCommand_Set except that it retrieves setting from scanner. This is a synchronized function, it returns when command has been sent to

and got response from scanner. USI_WaitForSendEchoTO is not needed before a next continuous send command call.

BOOL HAM_SendCommand_Get(WORD len, BYTE cmdID, BYTE op, BYTE* get, DWORD timeout)

Parameter: (input)

len:	BYTE	: Specifies length of command, which is actually always be 4.
cmdID:	BYTE	: command ID
op:	BYTE	: operation mode for this command
get:	BYTE*	: Pointer to a byte which will hold the setting retrieved from scanner.
timeout:	DWORD	: Specifies the timeout in millisecond

Return:

TRUE = indicates the setting has been retrieve successfully.

2.28 Send scanner command set string to decoding chip

Function Description:

This function call has the same function as HAM_SendCommand_Set except that it sends a serial of data instead of a single byte to scanner.

BOOL HAM_SendCommand_SetString(WORD len, BYTE cmdID, BYTE op, LPCSTR sets, int slen, DWORD timeout)

Parameter: (input)

len:	BYTE	: Specifies length of command, which will be calculated and adjusted automatically
cmdID:	BYTE	: command ID
op:	BYTE	: operation mode for this command
sets:	LPCSTR	: Specifies the string data
slen:	int	: Specifies the length of string data. Set to -1 to calculate automatically
timeout:	DWORD	: Specifies the timeout in millisecond

Return:

TRUE = indicates the setting has been set successfully.

2.29 Get scanner command set string from decoding chip

Function Description:

This function call has the similar function as HAM_SendCommand_SetString except that it retrieves setting from scanner

BOOL HAM_SendCommand_GetString(WORD len, BYTE cmdID, BYTE op, LPSTR gets, int* slen, DWORD timeout)

Parameter: (input)

len:	BYTE	: Specifies length of command, which will be calculated and adjusted automatically
cmdID:	BYTE	: command ID
op:	BYTE	: operation mode for this command
gets:	LPSTR	: Buffer which will hold the setting retrieved from scanner
slen:	int*	: Specifies the length of buffer and returns actual data length in the buffer
timeout:	DWORD	: Specifies the timeout in millisecond

Return:

TRUE = indicates the setting has been retrieve successfully.

2.30 Get scanner related version information

Function Description:

Get Scanner related version information. It does not need to call USI_Register to use this function.

Function call:

```
BOOL USI_GetScannerVersion(LPTSTR model, LPTSTR firmware, LPTSTR sdk, int blen);
```

Parameter: (output)

model:	LPTSTR	: scanner model.
firmware:	LPTSTR	: firmware version number.
sdk:	LPTSTR	: sdk version number if available.
blen:	int	: specifies buffer length for parameters of model, firmware and sdk.

Return:

Always True.

2.31 Enable prompt warming message from USI

Function Description:

Enables USI to report working information in a popup window

Function call:

```
BOOL USI_EnablePromptMessage(BOOL enable);
```

Parameter: (output)

enable:	BOOL	: True= enable, Fail:Disable

Return:

Always True.

2.32 Scanner working mode (available for 2D model)

Function Description:

Sets scanner engine to working mode of barcode decoding/Image/Preview (mode = SWM_BARCODE) or image capture (mode = SWM_IMAGE) or preview and image capture (mode = SWM_IMAGE_PREVIEW) for 2D scanner

Function call:

```
BOOL USI_SetWorkingMode(int mode);
```

Parameter: (output)

mode:	int	: mode = SWM_BARCODE – Barcode mode = SWM_IMAGE – image capture mode = SWM_IMAGE_PREVIEW – preview and image capture for 2D scanner

Return:

Always True.

2.33 Get image (available for 2D model)

Function Description:

Retrieves captured image in bitmap format, and returns image size.

Function call:

```
HBITMAP USI_GetImageBitmap(SIZE* imagesize);
```

Parameter: (output)

imagesize:	SIZE	: Bitmap image size

Return:

HBITMAP	: image

2.34 Resize image (available for 2D model)

Function Description:

Resizes a bitmap.

Function call:

HBITMAP USI_ResizeBitmap(HBITMAP hbmp, int cx, int cy, BOOL keepratio);

Parameter: (input)

hbmp:	HBITMAP	: the bitmap handle needs to be resized.
cx:	int:	: cx define new width dimension of the bmp.
cy	int:	: cy define new height dimension of the bmp.
keepratio:	BOOL:	: If keepratio is true, it will resize the bmp to fit the cx-cy rectangle and keep the ratio of the original image.

Return:

HBITMAP : image

2.35 Save image to file (available for 2D model)

Function Description:

Saves captured image to a file. The image format is determined by the file extension name, which can be .bmp, .jpg, .jpeg, .tif, .tiff or .raw. The compressionfactor is used for jpeg format.

Function call:

BOOL USI_SaveImageToFile(LPCTSTR filename, int compressionfactor);

Parameter: (input)

filename:	LPCTSTR	: File extension name defines format. HHP, SSI support .bmp, .jpg, and .tif. Tohken supports .bmp and .jpg.
compressionfactor:	int	: compressionfactor is used for when format is jpg.

Return:

Always True.

2.36 Get terminator

Function Description:

Returns terminator. The returned constant value is defined in USI_SetTerminator.

Function call:

int USI_GetTerminator();

Return:

Int : terminator

2.37 Set terminator

Function Description:

Sets terminator.

Function call:

BOOL USI_SetTerminator(int terminator);

Parameter: (input)

terminator:	int	: TERMINATOR_ENTER	: enter (CR/LF)
		TERMINATOR_RETURN	: return (CR)
		TERMINATOR_LINEFEED	: linefeed (LF)
		TERMINATOR_NONE	: no terminator
		TERMINATOR_ENTERENTER	: double enter (CR/CR)

Return:

Always True.

2.38 Get good read sound mode and sound name

Function Description:

Returns Good-Read-Echo mode and sound file name.

Function call:

int USI_GetGoodReadEcho(LPTSTR buffer, UINT blen);

Parameter: (input)

buffer:	LPTSTR	: returns sound file name which included path
blen:	UINT	: defines length of the buffer.

Return:

Int:	GRE_PLAYSOUND	: Play pre-set sound file
	GRE_BEEP	: Play default beep sound
	GRE_NONE	: No sound.

2.39 Set good read sound mode and sound name

Function Description:

Sets Good-Read-Echo mode and sound file name.

Function call:

BOOL USI_SetGoodReadEcho(int mode, LPTSTR SoundFileName);

Parameter: (input)

mode:	int	: GRE_PLAYSOUND	: Play pre-set sound file
		GRE_BEEP	: Play default beep sound
		GRE_NONE	: No sound.
buffer:	LPTSTR	: sound file name which included path	

Return:

It returns true when successful, false if mode is n/a.

2.40 Set previewsize (only for 2D engine)

Function Description:

Defines how large is the window for image preview.

Function call:

void USI_SetPreviewSize (SIZE size);

Parameter: (input)

size:	SIZE	: Size of image preview window

2.41 Set previewsize time-out (only for 2D engine)

Function Description:

Set timeout for preview in seconds. When in taking image preview mode, this timeout will stop preview and trigger to capture the image.

Function call:

void USI_SetPreviewTimeout (DWORD timeout);

Parameter: (input)

timeout:	DWORD	: timeout in seconds

2.42 Set good read sound mode and sound name

Function Description:

Sets Good-Read-Echo mode and sound file name.

Function call:

BOOL USI_SetGoodReadEcho(int mode, LPTSTR SoundFileName);

Parameter: (input)

mode:	int	: GRE_PLAYSOUND	: Play pre-set sound file
		GRE_BEEP	: Play default beep sound
		GRE_NONE	: No sound.
buffer:	LPTSTR	: sound file name which included path	

Return:

It returns true when successful, false if mode is n/a.

2.43 2D imager supporting for PA966/967

2D supporting API is described on individual document. Please get it from
http://w3.tw.ute.com/pub/cs/manual/WinCE_programming_manual/2D_Engine_SDK.pdf

3 Control command for decoder chip (Hamser: 1D only)

Important: This chapter describes low level command for scanner control function. If you already USI to do scanner programming, you don't need to care about this chapter. In general, it is not suggested to use level command to control scanner, because there are timing issue on serial communication programming , and it is always need communication expert to do that and it is hard to explain it on document.

When Host prepare to send a command to hamster, it must first check CTS, if CTS is high, then Host must set the RTS to high then clear RTS to low to wake up the Hamster.

Special Command for control		
command	Format	Comment
Control	Esc,80H+SOH(01H)	Let Hamster enter slaving status. At this status Hamster just receives commands and executes it until it receives Release command or timeout (about 10s). Otherwise, the timeout is about 1s as the interval of commands.
Release	Esc,80H+EOT(04H)	Let Hamster exit from slaving status.
Execute/ Enquiry	Esc,80H+ENQ(05H)	Let Hamster execute the previous saved command and check hamster if there is a result of previous executed command to send to Host. If previous saved command have already executed and no result to send, hamster do not reply until there is a result. If Host receive a result but the BCC is wrong, it can re-send ENQ to re-send result again.
ACK	Esc,80H+ACK(06H)	It is from Hamster to Host. If Hamster receive a command and this command do not need send message back, Hamster reply the ACK.
NAK	Esc,80H+NAK(15H)	It is from Hamster to Host. Hamster require the Host to re-send command again, normally when received a wrong BCC, it can send the NAK. The Hamster sends back NAK whenever it receives a no sense command.

COMMAND FROM HOST TO HMASTER

Command format:Esc,Lh,Ll,n,m,S1,...,Si,BCC
Here: **Esc** is Escape code(H'1B)
Lh/Ll is command's length when the Lh.b7 is 0, Lh is high byte, Ll is low byte, count from n to BCC. When Lh.b7=1 it is a two bytes special command.
n is command ID
m is operation: Normally for setting commands the 0 means setting, 1 default, 2 read current setting, 3 special operation. When m=1 or 2, the S1 should be 0 for bits or one character setting. If the setting is a string, like pre_amble, the read or default command should not contain any Si byte. The special meaning in a command please refers the command definition.
Si is setting/read data.
BCC: it equals to XOR of all the bytes before the BCC.

Conventions: S1.bj means the number j bit of byte S1.
The expression 1~64:2 means that the number is between 1 and 64, the default is 2.
Notice: Any interval in a command transmit can not exceed 1 second.

Command	Format	Comment
Initial/ Warm start	Esc,0,2,0,BCC	Hamster initializes the ports and flags according to the setting in RAM.
Default	Esc,0,2,1,BCC	Reset setting in RAM and initialize
Mpu_idle	Esc,0,4,2,m,S1,BCC	S1 is 0~3:0 is sleep mode,1 is watch mode, 2_is standby mode.
Beep	Esc,0,4,3,m,S1,BCC	S1 0 none,1 low,2_medium,3 high,4 low/high,5high/low
block_delay	Esc,0,4,4,m,S1,BCC	S1 is 0_10ms,1 50ms,2 100ms,3 500ms,4 1s,5 3s
char_delay	Esc,0,4,5,m,S1,BCC	S1 is 0_none,1 1ms,2 5ms,3 10ms,4 20ms,5 50ms
Function_code	Esc,0,4,6,m,S1,BCC No meaning for you	S1 is 0 off,1_on
Capslock	Esc,0,4,7,m,S1,BCC No meaning for you	S1 is 0_auto trace,1 lower case,2 upper case
Language	Esc,0,4,8,m,S1,BCC No meaning for you	S1 is 0_U.S.,1 U.K.,2 Swiss,3 Swedish, 4 Spanish,5 Norwegian,6 Italian,7 German,8 French,9 Alt Key Mode,A Danish
Baud_rate	Esc,0,4,0D,m,S1,BCC No meaning for you	S1 is 0 300,1 600,2 1200,3 2400,4 4800, 5 9600,6 19200,7_38400
Parity	Esc,0,4,0E,m,S1,BCC No meaning for you	S1 is 0 EVEN,1 ODD,2 MARK,3 SPACE,4_NONE
Data_bits	Esc,0,4,0F,m,S1,BCC No meaning for you	S1 is 0 7,1_8BIT
Handshake	Esc,0,4,10,m,S1,BCC No meaning for you	S1 is 0_IGNORE,1 RTS ENABLE AT POWERUP,2 RTS ENABLE IN COMMUNICATION
Ack_nak	Esc,0,4,11,m,S1,BCC No meaning for you	S1 is 0_OFF,1 ON
BCC_char	Esc,0,4,12,m,S1,BCC No meaning for you	S1 is 0_OFF,1 ON
Data_direction	Esc,0,4,13,m,S1,BCC No meaning for you	S1 is =0_SEND TO HOST,1 SEND TO HOST AND TERMINAL,2 SEND TO TERMINAL
Time_out	Esc,0,4,14,m,S1,BCC No meaning for you	S1 is 0_1S,1 3S,2 10S,3 UNLIMITED
Terminator	Esc,0,4,15,m,S1,BCC	S1 is B1B0=0_ENTER(CR/LF),1 FIELD EXIT(CR),2 RETURN(LF),3 NONE
Code_id	Esc,0,4,16,m,S1,BCC	S1 is 0_OFF,1 ON
Verification	Esc,0,4,17,m,S1,BCC	S1 is 0_OFF,1~7 1 to 7 times verification
Scan_mode	Esc,0,4,18,m,S1,BCC	S1 is 0_TRIGGER MODE,1 FLASH_MODE,2 MULTISCAN MODE,3 ONE PRESS ONE SCAN,4~7 reserved
Label_type	Esc,0,4,19,m,S1,BCC	S1 is 0_POSITIVE,1 POSITIVE AND NEGATIVE

Aim_fuction	Esc,0,4,1a,m,S1,BCC	S1 is 0_DISABLE,1 ENABLE
Scan_pre_data	Esc,0,L,1b,m,S1,…Si,BCC	Si can be 1 to 8 CHARACTERS
Scan_post_data	Esc,0,L,1c,m,S1,…Si,BCC	Si can be 1 to 8 CHARACTERS
Define_code39f	Esc,0,4,1d,m,S1,BCC	define Code 39 full ASCII ID:Here S1 is 1 CHARACTER
Define_code39s	Esc,0,4,1e,m,S1,BCC	define Code 39 standard ID:Here S1 is 1 CHARACTER
Define_EAN13	Esc,0,4,1f,m,S1,BCC	define EAN13 ID:Here S1 is 1 CHARACTER
Define_UPCA	Esc,0,4,20,m,S1,BCC	define UPC A ID: Here S1 is 1 CHARACTER
Define_EAN8	Esc,0,4,21,m,S1,BCC	define EAN8 ID:Here S1 is 1 CHARACTER
Define_UPCE	Esc,0,4,22,m,S1,BCC	define UPC E ID:Here S1 is 1 CHARACTER
Define_I25	Esc,0,4,23,m,S1,BCC	define I25 ID:Here S1 is 1 CHARACTER
Define_CDB	Esc,0,4,24,m,S1,BCC	define Codabar ID:Here S1 is 1 CHARACTER
Define_C128	Esc,0,4,25,m,S1,BCC	define Code128 ID:Here S1 is 1 CHARACTER
Define_C93	Esc,0,4,26,m,S1,BCC	define Code93 ID:Here S1 is 1 CHARACTER
Define_S25	Esc,0,4,27,m,S1,BCC	define S25 ID:Here S1 is 1 CHARACTER
Define_MSI	Esc,0,4,28,m,S1,BCC	define MSI ID:Here S1 is 1 CHARACTER
Define_C11	Esc,0,4,29,m,S1,BCC	define Code11 ID:Here S1 is 1 CHARACTER
Define_C32	Esc,0,4,2a,m,S1,BCC	define Code32 ID:Here S1 is 1 CHARACTER
Define_DELTA	Esc,0,4,2b,m,S1,BCC	define Delta ID:Here S1 is 1 CHARACTER
Define_LABEL	Esc,0,4,2c,m,S1,BCC	define Label code ID:Here S1 is 1 CHARACTER
Define_PLESSEY	Esc,0,4,2d,m,S1,BCC	define Plessey ID:Here S1 is 1 CHARACTER
Define_TELEPEN	Esc,0,4,2e,m,S1,BCC	define Telepen ID:Here S1 is 1 CHARACTER
Define_TOSHIBA /Matrix 25, China Postal	Esc,0,4,2f,m,S1,BCC	define Toshiba ID:Here S1 is 1 CHARACTER
Define_EAN128	Esc,0,4,30,m,S1,BCC	define EAN128 ID:Here S1 is 1 CHARACTER;IF H'FF, THEN USE "]C1"
Mterminator	Esc,0,4,31,m,S1,BCC No meaning for you	Here S1 is 0_ENTER,1 NONE
Sentinal	Esc,0,4,32,m,S1,BCC No meaning for you	S1 is 0 not send,1 send
Track_selection	Esc,0,4,33,m,S1,BCC No meaning for you	Here S1 is =0_ALL TRACKS,1 TRACK1 AND TRACK2,2 TRACK1 AND TRACK3,3 TRACK2 AND TRACK3,4 TRACK1,5 TRACK2,6 TRACK3
T2_account_only	Esc,0,4,34,m,S1,BCC No meaning for you	S1 is 0_NO,1 YES
Separator	Esc,0,4,35,m,S1,BCC No meaning for you	S1 is 1 CHARACTER
Must_have_data	Esc,0,4,36,m,S1,BCC No meaning for you	S1 is 0 YES,1_NO
Track1_sequence	Esc,0,L,37,m,S1,…Si,BCC No meaning for you	Si can be 1 to 16 CHARACTERS
Track2_sequence	Esc,0,L,38,m,S1,…Si,BCC No meaning for you	Si can be 1 to 8 CHARACTERS
Code39_set	Esc,0,4,39,m,S1,BCC	S1.B0 is for Code39_enable,S1.B1 is for Code39_standard,S1.B3B2 for Code39_cd,S1.B4 Code39_ss
Code39_enable	Esc,0,4,3a,m,S1,BCC	S1 is 0 disable,1_enable
Code39_sandard	Esc,0,4,3b,m,S1,BCC	S1 is 0_full ASCII,1 standard
Code39_cd:	Esc,0,4,3c,m,S1,BCC	S1 is 0 calculate&send,1 calculate&not send,2_not calculate
Code39_ss	Esc,0,4,3d,m,S1,BCC	Here S1 is 0 SS send,1_SS not send
Code39_min	Esc,0,4,3e,m,S1,BCC	S1 is 0~48:0 (min<=data len)
Code39_max	Esc,0,4,3f,m,S1,BCC	S1 is 0~48:48 (data len<=max)
I2of5_set	Esc,0,4,40,m,S1,BCC	S1 is S1.B0 is for I2of5_enable,S1.B1 is for I2of5_fixlength,S1.B3B2 is for

		I2of5_cd,S1.B5B4 is for I2of5_ss
I2of5_enable	Esc,0,4,41,m,S1,BCC	S1 is =0 disable,1_enable
I2of5_fixlength	Esc,0,4,42,m,S1,BCC	S1 is =0 on,1_off (record first 3 record len)
I2of5_cd	Esc,0,4,43,m,S1,BCC	S1 is =0 calculate&send,1 calculate&not send,2_no calculation
I2of5_ss	Esc,0,4,44,m,S1,BCC	S1 is 0 first digit suppressed,1 last digit suppressed,2_not supressed
I25_min	Esc,0,4,45,m,S1,BCC	S1 is 2~64:10 (min<=data len)
I25_max	Esc,0,4,46,m,S1,BCC	S1 is 2~64:64 (data len<=max)
S2of5_set	Esc,0,4,47,m,S1,BCC	S1 is S1.b0 is for S2of5_enable,S1.b1 is for S2of5_fixlength,S1.b3b2 is for S2of5_cd
S2of5_enable	Esc,0,4,48,m,S1,BCC	S1 is 0_disable,1 enable
S2of5_fixlength	Esc,0,4,49,m,S1,BCC	S1 is 0_on,1 off (record first 3 record len)
S2of5_cd	Esc,0,4,4a,m,S1,BCC	S1 is 0 calculate&send,1 calculate&not send, 2_not calculate
S25_min	Esc,0,4,4b,m,S1,BCC	S1 is 1~48:4 (min<=data len)
S25_max	Esc,0,4,4c,m,S1,BCC	S1 is 1~48:48 (data len<=max)
Code32_set	Esc,0,4,4d,m,S1,BCC	S1 is S1.b0 is for Code32_enable,S1.b1 is for Code32_sc,S1.b2 is for Code32_lc
Code32_enable	Esc,0,4,4e,m,S1,BCC	S1 is 0_disable,1 enable
Code32_sc	Esc,0,4,4f,m,S1,BCC	S1 is 0_leading char send,1 not send
Code32_lc	Esc,0,4,50,m,S1,BCC	S1 is 0_tailing char send,1 not send
Telepen	Esc,0,4,51,m,S1,BCC	S1 is S1.b0 is for Telepen_enable,S1.b1 is for Telepen_charset
Telepen_enable	Esc,0,4,52,m,S1,BCC	S1 is 0_disable,1 enable
Telepen_charset	Esc,0,4,53,m,S1,BCC	S1 is 0_standard,1 numeric
Ean128	Esc,0,4,54,m,S1,BCC	S1 is S1.b0 is for Ean128_id, S1.b1 is for Ean128_id
Ean128_enable	Esc,0,4,55,m,S1,BCC	S1 is 0 disable,1_enable
Ean128_id	Esc,0,4,56,m,S1,BCC	S1 is 0 ID disable,1_ID enable
Ean128_func1	Esc,0,4,57,m,S1,BCC	S1 is 1 char
Code128	Esc,0,4,58,m,S1,BCC	S1 is 0 disable,1_enable
Code128_min	Esc,0,4,59,m,S1,BCC	S1 is 1~64:1 (min<=data len)
Code128_max	Esc,0,4,5a,m,S1,BCC	S1 is 1~64:64 (data len<=max)
Msi_pleasey	Esc,0,4,5b,m,S1,BCC	S1 is S1.b0 is for Msi_p_enable,S1.b1 is for Msi_pleasey_cd, S1.b3b2 is for Msi_p_cdmode
Msi_p_enable	Esc,0,4,5c,m,S1,BCC	S1 is 0_disable,1 enable
Msi_pleasey_cd	Esc,0,4,5d,m,S1,BCC	S1 is 0 check digit send,1_not send
Msi_p_cdmode	Esc,0,4,5e,m,S1,BCC	S1 is 0 check digit double module 10,1 check digit module 11 plus 10,2 check digit single module 10
Msi_pleasey_min	Esc,0,4,5f,m,S1,BCC	S1 is 1~64:1 (min<=data len)
Msi_pleasey_max	Esc,0,4,60,m,S1,BCC	S1 is 1~64:64 (data len<=max)
Code93	Esc,0,4,61,m,S1,BCC	S1 is 0 disable,1_enable
Code93_min	Esc,0,4,62,m,S1,BCC	S1 is 1~48:1 (min<=data len)
Code93_max	Esc,0,4,63,m,S1,BCC	S1 is 1~48:48 (data len<=max)
Code11	Esc,0,4,64,m,S1,BCC	S1 is S1.b0 is for Code11_enable,S1.b1 is for Code11_cdnumber,S1.b2 Code11_cdsend
Code11_enable	Esc,0,4,65,m,S1,BCC	S1 is 0_disable, 1 enable
Code11_cdnumber	Esc,0,4,66,m,S1,BCC	S1 is 0 one check digit,1_two check digits
Code11_cdsend	Esc,0,4,67,m,S1,BCC	S1 is 0 check digit send,1_not send
Code11_min	Esc,0,4,68,m,S1,BCC	S1 is 1~48:1 (min<=data len)
Code11_max	Esc,0,4,69,m,S1,BCC	S1 is 1~48:48 (data len<=max)

Codabar_set	Esc,0,4,6a,m,S1,BCC	S1 is S1.b0 is for Codabar_enable, S1.b1 is for Codabar_ss, S1.b3b2 is for Codabar_cd, S1.b4 is for Codabar_CLSI
Codabar_enable	Esc,0,4,6b,m,S1,BCC	S1 is 0_disable,1 enable
Codabar_ss	Esc,0,4,6c,m,S1,BCC	S1 is 0 start&stop char send,1_not send
Codabar_cd	Esc,0,4,6d,m,S1,BCC	S1 is 0 check digit calculate&send,1 check digit calculate but not send,2_check digit not calculate
Codabar_CLSI	Esc,0,4,6e,m,S1,BCC	S1 is 0 CLSI format on,1_off
Codabar_min	Esc,0,4,6f,m,S1,BCC	S1 is 3~48:3 (min<=data len)
Codabar_max	Esc,0,4,70,m,S1,BCC	S1 is 3~48:48
Label_code	Esc,0,4,71,m,S1,BCC	S1 is S1.b0 is for Label_c_enable,S1.b1 is for Label_code_cd
Label_c_enable	Esc,0,4,72,m,S1,BCC	S1 is 0_disable,1 enable
Label_code_cd	Esc,0,4,73,m,S1,BCC	S1 is 0 check digit send,1 not send
Upc_a_set	Esc,0,4,74,m,S1,BCC	S1 is S1.b0 is for Upc_a_enable,S1.b1 is for Upc_a_ld,S1.b2 is for Upc_a_cd
Upc_a_enable	Esc,0,4,75,m,S1,BCC	S1 is 0 disable,1_enable
Upc_a_ld	Esc,0,4,76,m,S1,BCC	S1 is 0_leading digit send,1 not send
Upc_a_cd	Esc,0,4,77,m,S1,BCC	S1 is 0_check digit send,1 not send
Upc_e_set	Esc,0,4,78,m,S1,BCC	S1 is S1.b1 is for Upc_e_enable,S1.b2 is for Upc_e_ld,S1.b3 is for Upc_e_cd,S1.b4 is for Upc_e_expand,S1.b0 is for Upc_e_nsc
Upc_e_enable	Esc,0,4,79,m,S1,BCC	S1 is 0 disable,1_enable
Upc_e_ld	Esc,0,4,7a,m,S1,BCC	S1 is 0_leading digit send,1 not send
Upc_e_cd	Esc,0,4,7b,m,S1,BCC	S1 is 0 check digit send,1_not send
Upc_e_expand	Esc,0,4,7c,m,S1,BCC	S1 is 0 zero expansion on,1_off
Upc_e_nsc	Esc,0,4,7d,m,S1,BCC	S1 is 0_disable,1 enable
Ean_13_set	Esc,0,4,7e,m,S1,BCC	S1 is S1.b0 is for Ean_13_enable,S1.b1 is for Ean_13_ld,S1.b2 is for Ean_13_cd,S1.b3 is for Ean_13_bookland
Ean_13_enable	Esc,0,4,7f,m,S1,BCC	S1 is 0 disable,1_enable
Ean_13_ld	Esc,0,4,80,m,S1,BCC	S1 is 0_leading digit send,1 not send
Ean_13_cd	Esc,0,4,81,m,S1,BCC	S1 is 0_check digit send,1 not send
Ean_13_bookland	Esc,0,4,82,m,S1,BCC	S1 is 0 bookland EAN enable,1_ disable
Ean_8_set	Esc,0,4,83,m,S1,BCC	S1 is S1.b0 is for Ean_8_enable,S1.b1 is for Ean_8_ld,S1.b2 is for Ean_8_cd
Ean_8_enable	Esc,0,4,84,m,S1,BCC	S1 is 0 disable,1_enable
Ean_8_ld	Esc,0,4,85,m,S1,BCC	S1 is 0_leading digit send,1 not send
Ean_8_cd	Esc,0,4,86,m,S1,BCC	S1 is 0_check digit send,1 not send
Supplement_set	Esc,0,4,87,m,S1,BCC	S1 is S1.b0 is for Supplement_two, s1.b1 is for Supplement_five,S1.b2 is for Supplement_mh, S1.b3 is for Supplement_ssi.
Supplement_two	Esc,0,4,88,m,S1,BCC	S1 is 0_off,1 on
Supplement_five	Esc,0,4,89,m,S1,BCC	S1 is 0_off,1 on
Supplement_mh	Esc,0,4,8a,m,S1,BCC	S1 is 0_transmit if present,1 must present
Supplement_ssi	Esc,0,4,8b,m,S1,BCC	S1 is 0 Space been inserted, 1_Space not been inserted
Delta_code_set	Esc,0,4,8c,m,S1,BCC	S1 is S1.b0 is for Delta_c_enable,S1.b1 is for Delta_code_cdc,S1.b2 is for Delta_code_cds
Delta_c_enable	Esc,0,4,8d,m,S1,BCC	S1 is 0_disable,1 enable
Delta_code_cdc	Esc,0,4,8e,m,S1,BCC	S1 is 0_check digit calculate,1 not calculate
Delta_code_cds	Esc,0,4,8f,m,S1,BCC	S1 is =0 check digit send,1_not send
Get_version	Esc,0,3,90,2,BCC	Get firmware version.
DumpSetting	Esc,Lh,Ll,91,m,S1..	Lh/Ll is command length. Si is in the range of

	.Si,BCC	s1 to S255.m=0 is download setting, m=1 is reset the setting area into FF. m=2 is upload setting. Actually you just need the format as bellow: Download: Esc,1,02,91,0,s1,...,s255,BCC Upload: Esc,0,3,91,2,BCC
EAN128Brace Remove	Esc,0,4,92,m,S1,BCC	S1 is =0_disable,1 enable(Remove the brace)
AimingTime	Esc,0,4,93,m,S1,BCC	S1 is =0 0.5s,1_1s,2 1.5s 3 2s
Exchange data	Esc,Lh,Ll,a3,S1,S2,….,Sn, BCC	• Expect Acknowledge (Esc,80H+ACK(06H)) • Exchange the data between the host and the ICC. • Expected return after issuing Execute/Enquiry command are: Esc,Lh,Ll,0xa3,AH,data,BCC Here: AH=0 Success =1 Timeout =2 No card present data: Response data and status word
Note: Hamster save these commands to buffer and do not execute until it receives an Execute command (Esc,ENQ). Hamster execute the command after receive an "Esc,ENQ" then send back a reply. The Max. Length of data is 264. The m and the reply define as following:		

DATA TO HOST FROM HAMSTER					
Data format: Code_number,Lh,Ll,string Here: The Lh/Ll is string length, Lh is high byte, Ll is low byte, The string length is excluded the Code_number and Lh/Ll. The string contains the Code ID, pre_amble, scanned data,post_amble, and terminator. Code_number is equal to following number plus H'80.					
0 Code 39 full ASCII	1 Code 39 standard or EDP Code			2 EAN 13	3 UPC A
4 EAN 8	5 UPC E	6 I25	7 Codabar	8 Code 128	9 Code 93
10 S25	11 MSI	12 EAN 128	13 Code 32	14 Delta	15 Label
16 Plessey	17 Code 11	18 Toshiba	19 reserved	20 Track 1	21 Track 2
22 Track 3	23 More than 1 track	24 reserved	25 RS232	26 reserved	27 reserved
28 reserved	29 reserved	30 reserved	31 reserved	32 reserved	33 reserved

4 *SysIOAPI.DLL*

This DLL provide hardware relative API for user to control scanner, LED, back-light and slot. API functions are provided through DLL to assist programmer to write application. . Two files are essential and provided in SDK, SysIOAPI.LIB and SysIOAPI.H.

The link includes the sample program for SysIOAPI.

http://w3.tw.ute.com/pub/cs/software/Sample_Program/SysIOAPI/SysIOAPISample.zip

Note : For programming, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide *.H and *.LIB for compiler for Windows Mobile OS), please refer to Chapter 5 for programming guide.

Note : Not all device support these functions.

4.1 Keypad Related Functions

4.1.1 Disable/enable power button

Function Description:

To enable / disable power button

Function call:

void DisablePowerButton(BOOL)

Parameter (Input)

TRUE = Disable power button.
FALSE = Enable power button.

4.1.2 Set keypad utility input mode

Function Description:

In terminal, there is a utility to emulate full alpha key input, called GetVK. The input mode can be switched by pressing “alpha” key, or by following function.

Function call:

void SetGetVKWorkingMode(int)

Parameter (input)

0 = hide the selection window.
1 = show lower case selection window.
2 = show upper case selection window.

4.1.3 Get keypad utility input mode

Function Description:

This function is used to check alpha key input mode.

Function call:

BYTE GetAlphaKeyWorkingMode(void);

Return code:

0 = **normal** .
1 = **lower case**.
2 = **upper case.**

4.1.4 Check Alpha key is pressing

Function Description:
This function is used to check if alpha key is pressed or not.

Function call:
BOOL GetKeypadAlphaStatus(void);

Return code:
TRUE = Alpha key is pressed.
FALSE = Alpha key is released.

4.1.5 Enable/Disable Alpha Key

Function Description:
This function is used to enable/disable Alpha key.
Function call:
void SetAlphaKeyDisable (BOOL bDisable)
Parameter (Input):
TRUE = Disable Alpha key.
FALSE = Enable Alpha key.

4.1.6 Check Alpha Key Status

Function Description:
This function is used to check Alpha key is enabled or not.
Function call:
BOOL GetAlphaKeyStatus (void)
Return code:
TRUE = Alpha key is enabled.
FALSE = Alpha key is disabled.

4.1.7 Check Function key status

Function Description:
This function is used to check if function key is enabled or not.

Function call:
BOOL GetFnKeyStatus(void);

Return code:
TRUE = Function key is enabled.
FALSE = Function key is enabled.

4.1.8 Enable/Disable Function key

Function Description:
This function is used to enabled/disable function key.

Function call:
void SetFnKeyDisable(BOOL bOff);

Return code:
TRUE = Disable function key.
FALSE = Enable function key.

4.1.9 Set Function Mode

Function Description:
This function is used set keypad in function mode or not.
Function call:
void SetFnKeyWorkingMode (BOOL bEnable)
Parameter (Input):
TRUE = Set keypad is function mode.
FALSE = Set keypad is normal mode.

4.1.10 Get Function Mode

Function Description:
This function is used get the keypad is function mode or not.
Function call:
BOOL GetFnKeyWorkingMode ()
Return code:
TRUE = Keypad is in function mode.
FALSE = Keypad is in normal mode.

4.1.11 Detect Function Key Status

Function Description:
This function is used get the function key is pressed or not.
Function call:
BOOL GetKeypadFnKeyStatus ()
Return code:
TRUE = Function key is pressed.
FALSE = Function key is released.

4.1.12 Check Start Key Status

Function Description:
This function is used to check Start key is enabled or not.
Function call:
BOOL GetStartKeyStatus (void)
Return code:
TRUE = Start key is enabled.
FALSE = Start key is disabled.

4.1.13 Enable/Disable Start Key

Function Description:
This function is used to enable/disable Start key.
Function call:
void SetStartKeyDisable (BOOL bDisable)
Parameter (Input):
TRUE = Disable Start key.
FALSE = Enable Start key.

4.1.14 Check Talk Key Status

Function Description:
This function is used to check Talk key is enabled or not.
Function call:
BOOL GetPhoneTalkKeyStatus (void)
Return code:
TRUE = Talk key is enabled.
FALSE = Talk key is disabled.

4.1.15 Enable/Disable Talk Key

Function Description:
This function is used to enable/disable Talk key.
Function call:
void SetPhoneTalkKeyDisable(BOOL bDisable)
Parameter (Input):
TRUE = Disable Talk key.
FALSE = Enable Talk key.

4.1.16 Check Phone End Key Status

Function Description:
This function is used to check phone end key is enabled or not.
Function call:
BOOL GetPhoneEndKeyStatus (void)
Return code:
TRUE = phone end key is enabled.
FALSE = phone end key is disabled.

4.1.17 Enable/Disable Phone End Key

Function Description:
This function is used to enable/disable phone end key.
Function call:
void SetPhoneEndKeyDisable (BOOL bDisable)
Parameter (Input):
TRUE = Disable phone end key.
FALSE = Enable phone end key.

4.1.18 Check Keypad Status

Function Description:
This function is used to check keypad is locked or not.
Function call:
BOOL GetKeypadLockStatus (void)
Return code:
TRUE = Keypad is locked.
FALSE = Keypad is not locked.

4.1.19 Lock/Unlock Keypad

Function Description:
This function is used to lock/unlock keypad.
Function call:
void SetKeypadLock (BOOL bLock)
Parameter (Input):
TRUE = Lock keypad.
FALSE = Unlock keypad.

4.2 Scanner Related Functions

To save power, the decoder IC is disabled when scanner is not in use. It can be enabled through USI functions. Following functions are meaningful only if decode IC is enabled.

4.2.1 Enable/Disable Scanner trigger key

Function Description:
This function enables/disables trigger keys.

Function call:
void EnableScannerTrigger(BOOL fOn)

Parameter (Input)

fON:	BOOL:	TRUE = enable trigger keys. FALSE = disable trigger keys.

4.2.2 Power on/off Scan Engine

Function Description:
This function is used to power on/off scan engine.

Function call:
void PowerOnScanner (BOOL fOn)

Parameter (Input)

fON:	BOOL:	TRUE = Power on scan engine. FALSE = Power off scan engine.

4.2.3 Turn on/off Scan Engine

Function Description:
This function emulates trigger keys to turn scan engine on or off. It functions even if trigger keys are disabled.

Function call:
void SetScannerOn(BOOL fON)

Parameter(Input)

fON:	BOOL:	TRUE = turn scan engine on. FALSE = turn scan engine off.

4.2.4 Get Trigger keys Status

Function Description:
This function returns enable/disable status of trigger keys.

Function call:
BOOL GetScannerTrigger(void)

Return code:
TRUE = trigger keys are enabled.
FALSE = trigger keys are disabled.

4.2.5 Get Scanner Status

Function Description:

This function returns the status of scan engine, or trigger key.

Function call:

BOOL GetScannerStatus(void)

Return code:

TRUE = scan engine is on, or trigger key is pressed.
FALSE = scan engine is off, or trigger key is released.

4.2.6 Check Trigger key is pressing

Function Description:

This function is used to check if left or right trigger key is pressed or not.

Function call:

BOOL TriggerKeyStatus(int key);

Parameter(Input)

key:	int:	LEFT_TRIGGER_KEY : left trigger key RIGHT_TRIGGER_KEY: right trigger key.

Return code:

TRUE = trigger is pressed.
FALSE = trigger is released.

Example:

```
#define kKeybdTriggerEventName          TEXT("KeybdTriggerChangeEvent")
#define kKeybdAlphaKeyEventName         TEXT("KBDAlphaKeyChangeEvent")
#define LEFT_TRIGGER_KEY    1
#define RIGHT_TRIGGER_KEY 2

gKeyEvents[0] = CreateEvent(NULL, TRUE, FALSE, kKeybdTriggerEventName);
gKeyEvents[1] = CreateEvent(NULL, TRUE, FALSE, kKeybdAlphaKeyEventName);

while (1)
{
        WaitForMultipleObjects(2, gKeyEvents, FALSE, INFINITE);

        TriggerKeyStatus(LEFT_TRIGGER_KEY);
        TriggerKeyStatus(RIGHT_TRIGGER_KEY);
}
```

4.3 LED related function

4.3.1 Turn On/Off Good Read LED

Function Description:

This function is used to turn on/off good read LED.

Function call:

void GoodReadLEDOn(BOOL fON)

Parameter(Input)

fON: BOOL: TRUE = turn on good read LED.
FALSE = turn off good read LED.

4.3.2 Turn On/Off Camera LED

Function Description:

This function is used to turn on/off camera LED.

Function call:

void CameraLEDOn (BOOL bOn)

Parameter (Input):

TRUE = Turn on the camera LED.
FALSE = Turn off the camera LED.

4.3.3 Turn On/Off Green LED

Function Description:

This function is used to turn on/off green LED.

Function call:

void GreenLEDOn (BOOL bOn)

Parameter (Input):

TRUE = Turn on the green LED.
FALSE = Turn off the green LED.

4.3.4 Turn On/Off Red LED

Function Description:

This function is used to turn on/off red LED.

Function call:

void RedLEDOn (BOOL bOn)

Parameter (Input):

TRUE = Turn on the red LED.
FALSE = Turn off the red LED.

4.4 Backlight related function

There are two backlight controls, screen backlight and keypad backlight. They are controlled separately. For screen backlight, you can adjust brightness of backlight also.

4.4.1 Screen Backlight Control

Function Description:

This function turns screen backlight on or off.

Function call:

void BacklightOn(BOOL fON)

Parameter(Input)

fON:	BOOL:	TRUE = turn on screen backlight. FALSE= turn off backlight.

4.4.2 Get Screen Backlight Status

Function Description:

This function returns the status of screen backlight.

Function call:

BOOL GetBacklightStatus(void)

Return code:

TRUE = screen backlight is on.
FALSE = screen backlight is off.

4.4.3 Keypad Backlight Control

Function Description:

This function turns keyoad backlight on or off.

Function call:

void KeypadLightOn(BOOL fON)

Parameter(Input)

fON:	BOOL:	TRUE = turn on keypad backlight. FALSE = turn off backlight.

4.4.4 Get Keypad Backlight Status

Function Description:

This function returns the status of keyoad backlight.

Function call:

BOOL GetKeypadLightStatus(void)

Return code:

TRUE = keypad backlight is on.
FALSE = keypad backlight is off.

4.4.5 Screen Backlight Brightness Control

Function Description:
This function adjusts screen backlight brightness.

Function call:
void BrightnessUp(BOOL fup)

Parameters(Input)

Fup:	BOOL:	TRUE = adjust one step up. FALSE = adjust one step down.

4.5 SD slot related functions

4.5.1 Inquire SD slot status

Function Description:
This function returns SD slot enable/disable status.

Function call:
BOOL GetSDStatus()

Return code:
TRUE = Slot is enabled.
FALSE = Slot is disabled.

4.5.2 Enable/Disable SD Slot

Function Description:
This function enables/disables SD slot.

Function call:
void EnableSDSlot(BOOL bEnable);

Parameters(Input)

bEnable:	BOOL:	TRUE = Enable slot. FALSE = Disable slot.

4.5.3 Get SD Slot Working Mode

Function Description:
This function is used to get SD slot mode.
Function call:
DWORD GetSDMode (void)
Return code:
0x01 : SDMMC mode.
0x02 : SDIO mode.
0x11 : SDMMC without remove while resuming.

4.5.4 Set SD Slot Working Mode

Function Description:
This function is used to setup SD slot working mode.
Function call:
void SetSDMode(DWORD nMode)
Parameters(Input)

nMode : UINT32:		0x01 : SDMMC mode. 0x02 : SDIO mode. 0x11 : SDMMC without remove while resuming.

4.6 Enable/Disable Vibration

Function Description:

This function enables/disables vibration.

Function call:

void VibrationOn(BOOL bEnable);

Parameters(Input)

bEnable:	BOOL:	TRUE = On. FALSE = Off.

4.7 Camera Auto Focus

Function Description:

This function calls camera to focus.

Function call:

BOOL TriggerAutoFocus(void);

Return Code:

TRUE = Auto focus success.
FALSE = Auto focus fail.

4.8 Device Information

4.8.1 Get Smart Battery ID

Function Description:

This function is used to get the smart battery ID.

Function call:

BYTE GetSmartBatteryID()

Return Code:

0x00/0x01 : Not support smart battery ID.
Others Value : The smart battery ID.

4.8.2 Check Cradle Status

Function Description:

This function is used to check the device is on cradle or not.

Function call:

BOOL GetCradleStatus()

Return Code:

TRUE	= Device is on cradle.
FALSE	= Device is not on cradle.

4.9 Microphone, Receiver and Speaker Control

4.9.1 Switch Receiver and Speaker

Function Description:

This function is used to switch sound output from receiver or speaker.

Function call:

void Sound_To_Speaker_Or_Receiver (BOOL fOn)

Parameters(Input)

fOn:	BOOL:	TRUE = ON. FALSE = OFF

4.9.2 Get Main Microphone

Function Description:

This function is used to get the main microphone.

Function call:

DWORD Get_Main_Mic()

Return code:

1 : The front side microphone.
2 : The back side microphone.

4.9.3 Set Main Microphone

Function Description:
This function is used to switch main microphone.
Function call:
void Set_Main_Mic(DWORD dwIndex)
Parameters(Input)
dwIndex: DWORD : 1 : The front side microphone.
2 : The back side microphone.

4.10 Wireless module related functions

4.10.1 Inquire wireless module status

Function Description:
This function returns wireless module enable/disable status.

Function call:
BOOL GetWLANStatus()

Return code:
TRUE = Module is enabled.
FALSE = Module is disabled.

4.10.2 Enable/Disable wireless module

Function Description:
This function enables/disables wireless module.

Function call:
void WLANPowerEnable(BOOL bOn);

Parameters(Input)

bOn:	BOOL:	TRUE = Enable module.
		FALSE = Disable module.

4.11 Bluetooth module related functions

4.11.1 Enable/Disable Bluetooth Power status

Function Description:
Enable Bluetooth Module Power ON/OFF

Function call:
void BT_PowerEnable (BOOL bEnable)

Parameter (Input)

bON:	BOOL:	TRUE = Enable
		FALSE = Disable

4.11.2 Get BT Power status

Function Description:
Get Bluetooth Module Power Status

Function call:
BYTE BT_PowerStatus (void)

Return code:

BYTE:	1 = Bluetooth Module is Power ON
	0 = Bluetooth Module is Power OFF

4.12 PCMCIA/CF slot related functions

In HT660, it only support CF slot and PA96x/PA982 can support both CF and PCMCIA slot. So, please note that PCMCIA function is not work on following API in this section.

4.12.1 Get physical slot ID

Function Description:

PA96x/PA982 has two PC card slots, slot 0 and slot 1, for PCMCIA and CF. this function return which slot for PCMCIA or CF

Function call:

UINT GetPCMCIASlotID(UINT)

Parameters(Input)

0 = PCMCIA. (For PA962/PA966/PA982 only)
1 = CF.

Return code:

Physical slot ID.

4.12.2 Enable/Disable PCMCIA or CF slot

Function Description:

This function enables/disables PCMCIA or CF slot. PA96x/PA982 assigns physical slot 0 to CF and slot1 to PCMCIA, which is reversed compared with previous products. The following function is kept for compatible reason. It takes the same uSocket value as previous products, but reversed internally.

Function call:

void EnablePCMCIASlot(UINT uSocket, BOOL bEnable)

Parameters(Input)

uSocket:	UINT:	0 = PCMCIA slot. ***(PA962/PA966/PA982 only)***
		1 = Compact flash slot.
bEnable :	BOOL:	TRUE = enable specified slot.
		FALSE = disable specified slot.

4.12.3 Enable/Disable IO slots

Function Description:

This function enables/disables IO slots. It is recommended to use with function GetPCMCIASlotID() for platform independent reason.

Function call:

void EnablePCMCIASlot1(UINT uSocket, BOOL bEnable)

Parameters(Input)

uSocket:	UINT:	slot to be applied.
bEnable :	BOOL:	TRUE = enable specified slot.
		FALSE = disable specified slot.

Example

To disable PCMCIA slot and enable CF slot,

```
#define PCMCIA_SOCKET       0 (PA966/PA962/PA982 only)
#define CF_SOCKET           1
EnablePCMCIASlot1(GetPCMCIASlotID(PCMCIA_SLOT),FALSE);
EnablePCMCIASlot1(GetPCMCIASlotID(CF_SLOT),TRUE);
```

4.12.4 Inquire PCMCIA/CF slot status

Function Description:

This function returns PCMCIA/CF slot enable/disable status. Terminal assigns physical slot 0 to CF and slot1 to PCMCIA, which is reversed compared with previous products. The following function is kept for compatible reason. It takes the same uSocket value as previous products, but reversed internally.

Function call:

BOOL GetPCMCIAStatus(UINT uSocket)

Parameters(Input)

uSocket:	UINT:	0 = PCMCIA slot. ***(PA966/PA962/PA982 only)***
		1 = Compact flash slot.

Return

bEnable :	BOOL:	TRUE = Slot is enabled.
		FALSE = Slot is disable.

4.12.5 Inquire IO slot status

Function Description:

This function returns slot enable/disable status. It is recommended to use with function GetPCMCIASlotID() for platform independent reason.

Function call:

BOOL GetPCMCIAStatus1(UINT uSocket)

Parameters(Input)

uSocket:	UINT:	slot to be applied.

Return

bEnable :	BOOL:	TRUE = specified slot is enabled.
		FALSE = specified slot is disable.

Example

To check PCMCIA slot status,

```
#define PCMCIA_SOCKET     0 (PA966/PA962/PA982 only)
#define CF_SOCKET         1

if (GetPCMCIAStatus1(GetPCMCIASlotID(PCMCIA_SLOT)))
{
}
```

4.12.6 Disable PCMCIA/CF slot when resume

Function Description:

This function will disable the specified slot after resume even though that slot is enabled before suspend.

Function call:

void DisablePCMCIAUponResume(UINT uSocket, BOOL bDisable);

Parameters(Input)

uSocket:	UINT:	1 = physical socket 1
		0 = for physical socket 0
bDisable:	BOOL:	TRUE disable on resume
		FALSE enable on resume

4.13 Enable/Disable LCD screen

Function Description:
Turn on / off LCD screen
Function call:
void PowerOnColorLCD(BOOL fON)
Parameters(Input)

fON:	BOOL:	TRUE = Power on LCD screen FALSE = Power off LCD screen

5 Dynamic Load DLL

Compiler would not load the DLL while use dynamic load DLL, it help user to load the DLL if it exists while the application executed. The follow is the example.

Note: Even user does not need include the header and lib file but need to know the function definition.

```
////////////////////////////////////////////////////////////////////////////////
//C++

HINSTANCE g_hUSIDLL;
typedef BOOL (*lpfnUSI_GetScannerVersion)(LPTSTR model, LPTSTR firmware, LPTSTR sdk, int blen);
lpfnUSI_GetScannerVersion USI_GetScannerVersion;

g_hUSIDLL = LoadLibrary(L"\\Windows\\USI.dll");

if (g_hUSIDLL != NULL)
{
USI_GetScannerVersion = (lpfnUSI_GetScannerVersion)GetProcAddress(g_hUSIDLL,
TEXT("USI_GetScannerVersion"));
}
else
{
MessageBox(_T("Load library USI.dll fail"), NULL, MB_OK);
return;
}

TCHAR lstrmodel[50], lstrfirmware[50], lstrsdk[50];

if (USI_GetScannerVersion != NULL)
rc = USI_GetScannerVersion(lstrmodel, lstrfirmware, lstrsdk, sizeof(lstrmodel) + sizeof(lstrfirmware) +
sizeof(lstrsdk));
else
    MessageBox(_T("USI_GetScanerVersion does not find"), NULL, MB_OK);

if (g_hUSIDLL != NULL)
            FreeLibrary(g_hUSIDLL);

////////////////////////////////////////////////////////////////////////////////
//C#
[DllImport("SysIOAPI.dll", EntryPoint = "VibrationOn")]
public static extern void VibrationOn(bool bEnable);
```

6 *RFID HF Reader*

To programming RFID HF reader, it needs C++ library. Please get it from below URL.
http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip
And get the sample program from the link.
http://w3.tw.ute.com/pub/cs/software/RFID/HF/RFID_HF.zip

6.1 General Function

6.1.1 Get library version

Function Description:
To get the library version.

Function Call:
INT32 RDINTsys_GetAPIVersionString (LPWSTR strVersion);

Parameter:
strVersion: Get the library version.

Return code:
Please refer to section 6.5.

6.1.2 Connect to RFID reader

Function Description:
To create a connection with the reader before control it.

Function Call:
INT32 RDINTsys_OpenReader (BYTE u8COMPort，UINT32 u32Baudrate，CONST LPTSTR strAccessCode，BYTE u1SecurityMode，UINT32 u32OpenDelayMs，PUINT32 pu32Baudrate)

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u32Baudrate:	The reader's baud rate and the defualt is 19200.It supports 9600, 19200, 38400 and 115200.
strAccessCode:	The reader's access code, default is "00000000"
u1SecurityMode:	To set use security mode or not. TURN_ON : Open. TURN_OFF : Close.
u32OpenDelayMs:	The delay time for wait reader initial, we suggest this value is 700.
pu32Baudrate:	Receive the current reader's baud rate, if it is NULL then it will not receive the value.

Return code:
Please refer to section 6.5.

6.1.3 Close Reader

Function Description:
To finish controlling the reader.

Function Call:
INT32 RDINTsys_CloseReader (BYTE u8COMPort);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)

Return code:
Please refer to section 6.5.

6.1.4 Select Card type

Function Description:

This API change the reader working type with different card type and this should be called before read the card.

Function Call:

INT32 RDINT_WorkingType (BYTE u8COMPort, BYTE u8Type);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Type:	WT_ISO14443_TypeA
	WT_ISO14443_TypeB
	WT_ISO15693
	WT_SR176_SRIX4K

Return code:

Please refer to section 6.5.

6.1.5 Get Reader Information

Function Description:

Get the reader's serial number and firmware version.

Function Call:

INT32 RDINTv2_ReaderInfo (BYTE u8COMPort, LPBYTE pu8SerialNum, LPBYTE pu8FirmwareVer);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
pu8SerialNum:	Get reader's serial number (Length: READER_SERIAL_LEN)
pu8FirmwareVer:	Get reader's firmware version (Length: FIRMWARE_VER_LEN)

Return code:

Please refer to section 6.5.

6.1.6 Antenna Control

Function Description:

Enable/Disable antenna to save power.

Function Call:

INT32 RDINTv2_AntennaControl (BYTE u8COMPort, BYTE u8Select);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)	
u8Select:	ANTENNA_SELECT_ON	Open the antenna
	ANTENNA_SELECT_OFF	Close the antenna
	ANTENNA_SELECT_AUTO_POWER_LOW	Auto turn to low power mode.
	ANTENNA_SELECT_POWER_LOW	Manually turn to low power mode.

Return code:

Please refer to section 6.5.

6.2 ISO-15693

6.2.1 Inventory

Function Description:

Set the card to StayQuiet mode and return the card ID.

Function Call:

INT32 RDINT_ISO15693Inventory(BYTE u8COMPort, BYTE u8Flag, BYTE u8Afi, BYTE u8MaskLen, LPBYTE pu8Mask, LPBYTE pu8Dsfid, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
u8Afi:	Application Family Identifier parameter, please refer to ISO15693 document
u8MaskLen:	The mask length indicates the number of significant bits of the mask value. It can have any value between 0 and 60 when 16 slots are used and any value between 0 and 64 when 1 slot is used. LSB shall be transmitted first.
pu8Mask:	The mask value is contained in an integer number of bytes. LSB shall be transmitted first.
pu8Dsfid:	The Data Storage Format Identifier parameter, please refer to ISO15693 document.
pu8Uid:	The point of the buffer which to receive the tag ID.

Return code:

Please refer to section 6.5.

6.2.2 Set StayQuiet Mode

Function Description:

Set the card to StayQuiet mode.

Function Call:

INT32 RDINT_ISO15693StayQuiet(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contain the tag ID.

Return code:

Please refer to section 6.5.

6.2.3 Set Select Mode

Function Description:

Set the card to Select mode.

Function Call:

INT32 RDINT_ISO15693Select(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.

Return code:

Please refer to section 6.5.

6.2.4 Set Ready Mode

Function Description:

Set the card to Ready mode for StayQuiet or Select mode.

Function Call:

INT32 RDINT_ISO15693Reset2Ready(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.

Return code:

Please refer to section 6.5.

6.2.5 Read The Block Data form ISO15693 Tag

Function Description:

Read the block data from the specific ISO15693 tag.

Function Call:

INT32 RDINT_ISO15693Read(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8BlockStart, BYTE u8BlockCount, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.
u8BlockStart:	The first block which you want to read(ex : 0, 1, 2…).
u8BlockCount:	The number of blocks which you want to read.
pu8Data:	The point of the buffer which receive the block data.

Return code:

Please refer to section 6.5.

6.2.6 Write The Block Data to ISO15693 Tag

Function Description:

Write the block data to the specific ISO15693 tag.

Function Call:

INT32 RDINT_ISO15693Write(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8Block, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.
u8Block:	The block which you want to write(ex : 0, 1, 2…).
pu8Data:	The point of the buffer which contains the data.

Return code:

Please refer to section 6.5.

6.2.7 ISO15693 Lock Block

Function Description:
Lock the block on ISO15693 tag.

Function Call:
INT32 RDINT_ISO15693LockBlock(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8Block);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.
u8Block:	The block which you want to write(ex : 0, 1, 2…).

Return code:
Please refer to section 6.5.

6.2.8 Write AFI to ISO15693 Tag

Function Description:
Write AFI to the specific ISO15693 tag.

Function Call:
INT32 RDINT_ISO15693WriteAfi(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8AfiValue);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.
u8AfiValue:	The value of AFI and about this value please refer to ISO15693 document.

Return code:
Please refer to section 6.5.

6.2.9 ISO15693 Lock AFI

Function Description:
Lock the AFI on ISO15693 tag.

Function Call:
INT32 RDINT_ISO15693LockAfi(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.

Return code:
Please refer to section 6.5.

6.2.10 Write DSFID to ISO15693 Tag

Function Description:

Write DSFID to the specific ISO15693 tag.

Function Call:

INT32 RDINT_ISO15693WriteDsfid(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8DsfidValue);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.
u8DsfidValue:	The value of DSFID and about this value please refer to ISO15693 document.

Return code:

Please refer to section 6.5.

6.2.11 ISO15693 Lock DSFID

Function Description:

Lock the DSFID on ISO15693 tag.

Function Call:

INT32 RDINT_ISO15693LockDsfid(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Flag:	It specifies the actions to be performed by the VICC and whether corresponding fields are present or not.
pu8Uid:	The point of the buffer which contains the tag ID.

Return code:

Please refer to section 6.5.

6.2.12 Get Data From Reader

Function Description:

Get the data from the reader's buffer.

Function Call:

INT32 RDINT_GetReturnDataArray (BYTE u8COMPort, BYTE u8Indx, BYTE u8Offset, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8Indx:	The start index of pu8Data.
u8Offset:	The data's length which get from ISO15693AutoInventory4Antennas
pu8Data:	The point which will receive the data from reader. The front four bytes mean which antenna get how many tags, the others are the data.

Return code:

Please refer to section 6.5.

6.3 ISO-14443A

6.3.1 Write Default Key

Function Description:

Write the default key to reader.

Function Call:

INT32 RDINT_WriteDefaultKey(BYTE u8COMPort, BYTE u8DefaultKeyIndx, LPBYTE pu8DefaultKey);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8DefaultKeyIndx:	The default key index in the reader
pu8DefaultKey:	The point of the buffer which contain key.

Return code:

Please refer to section 6.5.

6.3.2 ISO-14443A Open Card

Function Description:

Lock the ISO-14443A tag and get the tag ID. After select card type, user should call this API before control the ISO-14443A tag.

Function Call:

INT32 RDINT_OpenCard(BYTE u8COMPort, BYTE u1AutoFind, LPBYTE pu8Uid, LPBYTE pu8Atqa, LPBYTE pu8Sak);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u1AutoFind:	Auto search card. TURN_ON TURN_OFF
pu8Uid:	Receive the tag ID.
pu8Atqa:	Return the ATQA card type. ATQA_MIFAER_S50 ATQA_MIFAER_S70 ATQA_ULTRA_LIGHT
pu8Sak:	Return the SAK card type. SAK_ISO14443_3 SAK_ISO14443_4

Return code:

Please refer to section 6.5.

6.3.3 ISO-14443A Close Card

Function Description:

Unlock the ISO-14443A tag. After control the ISO-14443A tag, user should call this API to unlock the tag.

Function Call:

INT32 RDINT_CloseCard(BYTE u8COMPort);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)

Return code:

Please refer to section 6.5.

6.3.4 ISO-14443A Read Block Data

Function Description:

Read the specify block data.

Function Call:

INT32 RDINT_ReadMifareOneBlock(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Block, LPBYTE pu8Key, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u1KeyType:	The private key type. CARD_KEY_A CARD_KEY_B
u1DefaultKey:	Use the default password in the reader. TURN_ON TURN_OFF
u8DefaultKeyIndx:	The index of the default key which in the reader.
u8Block:	The block which you want to read (Ex : 0, 1, 2…).
pu8Key:	The user defines key value.
pu8Data:	Receive the data.

Return code:

Please refer to section 6.5.

6.3.5 ISO-14443A Read Sector Data

Function Description:

Read the specify sector data.

Function Call:

INT32 RDINT_ReadMifareOneSector(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Sector, LPBYTE pu8Key, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u1KeyType:	The private key type. CARD_KEY_A CARD_KEY_B
u1DefaultKey:	Use the default password in the reader. TURN_ON TURN_OFF
u8DefaultKeyIndx:	The index of the default key which in the reader.
u8Sector:	The sector which you want to read (Ex: 0, 1, 2…).
pu8Key:	The user defines key value.
pu8Data:	Receive the data.

Return code:

Please refer to section 6.5.

6.3.6 ISO-14443A Write Block Data

Function Description:

Write data to the specify block.

Function Call:

INT32 RDINT_WriteMifareOneBlock(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Block, LPBYTE pu8Key, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u1KeyType:	The private key type. CARD_KEY_A CARD_KEY_B
u1DefaultKey:	Use the default password in the reader. TURN_ON TURN_OFF
u8DefaultKeyIndx:	The index of the default key which in the reader.
u8Block:	The block which you want to read (Ex: 0, 1, 2…).
pu8Key:	The user defines key value.
pu8Data:	The data which user wants to write.

Return code:

Please refer to section 6.5.

6.4 ISO-14443B

6.4.1 Select ST Card

Function Description:
Select ST card.

Function Call:
INT32 RDINT_STCardSelect(BYTE u8COMPort, LPBYTE pu8IDNum);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
pu8IDNum:	The card data.

Return code:
Please refer to section 6.5.

6.4.2 Release ST Card

Function Description:
Release ST card.

Function Call:
INT32 RDINT_STCardIntoDeactive(BYTE u8COMPort);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)

Return code:
Please refer to section 6.5.

6.4.3 Read SR176 Card's Block Data

Function Description:
Read the block from specify SR176 card.

Function Call:
INT32 RDINT_SR176ReadBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8BlkNo:	The block number which user wants to read (Ex: 0, 1, 2…).
pu8Data:	Receive the data.

Return code:
Please refer to section 6.5.

6.4.4 Write SR176 Card's Block Data

Function Description:
Write data to specify SR176 card's block.

Function Call:
INT32 RDINT_SR176WriteBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8BlkNo:	The block number which user wants to read (Ex: 0, 1, 2…).
pu8Data:	The data which user wants to write to the block.

Return code:
Please refer to section 6.5.

6.4.5 Lock SR176 Block

Function Description:
Lock the specify block on SR176 card.

Function Call:
INT32 RDINT_SR176LockBlock(BYTE u8COMPort, BYTE u8BlkNo);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8BlkNo:	The block number which user wants to lock (Ex: 0, 1, 2…).

Return code:
Please refer to section 6.5.

6.4.6 Read SRIX4K Card's Block Data

Function Description:
Read the block from specify SRIX4K card.

Function Call:
INT32 RDINT_SRIX4KReadBlock (BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8BlkNo:	The block number which user wants to read (Ex: 0, 1, 2…).
pu8Data:	Receive the data.

Return code:
Please refer to section 6.5.

6.4.7 Write SRIX4K Card's Block Data

Function Description:
Write data to specify SRIX4K card's block.

Function Call:
INT32 RDINT_SR176WriteBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
u8BlkNo:	The block number which user wants to read (Ex: 0, 1, 2…).
pu8Data:	The data which user wants to write to the block.

Return code:
Please refer to section 6.5.

6.4.8 Authenticate SRIX4K Card

Function Description:
Authenticate the specify SRIX4K card.

Function Call:
INT32 RDINT_SRIX4KAuth(BYTE u8COMPort, LPBYTE pu8Auth);

Parameter:

u8COMPort:	The reader's COM port number (1 – 255)
pu8Auth:	The authentication data.

Return code:
Please refer to section 6.5.

6.4.9 Read SRIX4K Card ID

Function Description:

Read the SRIX4K card ID.

Function Call:

INT32 RDINT_SRIX4KReadUID(BYTE u8COMPort, LPBYTE pu8Uid);

Parameter:

u8COMPort: The reader's COM port number (1 – 255)
pu8Uid: Receive the card ID.

Return code:

Please refer to section 6.5.

6.5 Error Code

Name	Value	Description
LRSUCCESS	0x00	Successful completion of request
LRSYSTEM	0x01	Unknown error
LRLASTCARD	0x02	Last Card Still Present
LRNOCARD	0x03	Card is not present
LRCTYPE	0x04	Card Type error
LRPARAM	0x05	Request Parameter error
LRACCESS	0x06	Card access error
LRREAD	0x07	Card read error
LRWRITE	0x08	Card write error
LRINCR	0x09	Purse increment error
LRDECR	0x0A	Purse decrement error
LRTRANSFER	0x0B	Purse value transfer error
LRRESTORE	0x0C	Purse restore error
LRPURSE	0x0D	Purse value corrupt
LRMADERR	0x0E	Card Directory error
LRFIXERR	0x0F	Purse fix error
LRFIXED	0x10	Purse found corrupt but fixed
LRNOTOPEN	0x11	Card not open
LRNOFILE	0x12	File not found
LRBADSIZE	0x13	Bad file size
LRABORTED	0x14	Request aborted
LRMANYCARD	0x15	Too many card present
LRFORMAT	0x16	Card format error
LRCREATE	0x17	Card file create error
LRDELETE	0x18	Card file delete error
LRALREADOPEN	0x19	Card has been opened already
LRALREADCLOSED	0x1A	Card has been closed already
LRMSTRKEYLOAD	0x1B	Cannot load master keys
LRAPPKEYLOAD	0x1C	Cannot load application keys
LRKEYCARD	0x1D	Keycard Error
LRUNFORMAT	0x1E	Card has files on it
LRNOKBDCHAR	0x20	No keyboard character
LRAPIWRITE	0x21	API data write error
LRAPIREAD	0x22	API data read error
LRBLOCKERROR	0x23	Block number error
LRNOTIMPL	0x7F	Function not implemented
LRUNKNOWN	0x80	Unknown error
LRCCRBUSY	0xBB	Reader is busy
LRCHANGEWROKTYPE	0xEE	Change Reader working type error
LRRDUNKNOWN	0xEF	Reader Unknown error
LRCRDNOTOPEN	0xFA	Card has not been opened

LRINUSE	0xFB	Card in use by another applications
LRAPPLICERR	0xFC	API system error
LRLINKLOST	0xFD	Link to Reader has been lost
LRBADCOMPORT	0xFE	COM port cannot be accessed
LRNOINIT	0xFF	Reader has not been opened
LRNOCRYPTBOX	0xF9	Key-Box not found
LRBADAPPACCESS	0xF8	Invalid Application access code
LRBADRDACCESS	0xF7	Invalid Reader access code
LRNOMAIDFILE	0xF6	Cannot open MAID definition
LRBOXREAD	0xF5	Cannot read from Key-Box
LRBOXWRITE	0xF4	Cannot write to Key-Box
LRBOXNOKEYS	0xF3	No of Keys in Box is zero or invalid
LRSECURE	0xF2	Comms MAC checking failed
LRERRSELREADER	0xF1	Cannot change to selected reader
LRRDNORESP	0xF0	Reader No Response
LRSAMUNKNOWN	0xED	SAM card unknown error

7 UHF reader for WJ

To programming UHF reader for WJ, it need C# DLL “MPR DLL.dll”. Please get it from below URL.
http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip

7.1 Class “MPRReader”

This is the main class instantiated by Applications. Manages a single WJ Multi-Protocol Reader. Provides properties and methods for accessing features of the MPR. Talks to the MPR via an MPRComm object. Generates request frame payloads for MPR API commands. Parses response frame payloads from MPR API commands. Fires events when MPR public properties change.

7.2 The Parameter in MPRReader

- byte **ActiveAntenna**
  This parameter to set and get the active antenna on reader and this value should be 0 on RH767.
- byte **TxPower**
  This parameter to set and get the current antenna power and this value should between 18-30.
- int **InvUpdateGap**
  To set and get the time between two inventories. Please set this value to 0 to get good performance.
- TimeSpan **PersistTime**
  How long a tag that has been read will persist in the inventory, without being read, while an inventory is running. If an inventory is stopped, tags do not expire. If a tag is re-read, it will live at least another PersistTime.
- bool **Class0InventoryEnabled**
  Whether to perform EPC Class 0 inventories.
- bool **Class1InventoryEnabled**
  Whether to perform EPC Class 1 inventories.
- bool **Gen2InventoryEnabled**
  Whether to perform EPC UHF Gen2 inventories.
- bool **IsConnected**
  To check the connection with reader.
- bool **InvTimerEnabled**
  To get or set inventory status.

7.3 The Function in MPRReader

7.3.1 Connect to RFID Reader

Function Description:

To create a connection with the reader before control it.

Function Call:

bool Connect(string SerialPortName, string BaudRate)

Parameter:

SerialPortName: The reader's COM port number (COM1: – COM255:)
BaudRate: The baud rate with the reader, the default is "57600"

Return code:

True :	Connect success.
False :	Connect fail.

7.3.2 Disconnect with RFID Reader

Function Description:

Close the connection and disable inventory with reader.

Function Call:

void Disconnect()

7.3.3 Clear All Tags In The Reader

Function Description:

Remove all tags in the buffer of reader.

Function Call:

void ClearInventory()

7.3.4 The Event in MPRReader

- **EventHandler InvTimerEnabledChanged**
  Fired when manufacturing information is read from the reader.
- **TagEventHandler TagAdded**
  Fired when a new tag is added to the inventory.
- **TagEventHandler TagRemoved**
  Fired when a tag expires, i.e. hasn't been read for the persist time period.

8 UHF reader for SkyeTek

To programming UHF reader for SkyeTek, it need C++ DLL “Skyedll.dll”. Please get it from below URL.
http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip

8.1 Connect to RFID reader

Function Description:

To create a connection with the reader before control it.

Function Call:

BOOL OpenPort (int nPort, DWORD dwBaudRate);

Parameter:

nPort:	The reader’s COM port number (1 – 255), default is COM2.
dwBaudRate:	The baud rate with the reader, the default is “57600”

Return code:

TRUE:	Connect success.
FALSE:	Connect fail

8.2 Disconnect with RFID reader

Function Description:

Close the connection with reader.

Function Call:

void ClosePort ();

8.3 Select Tag

Function Description:

To select any type of supported tag.

Function Call:

BOOL SelectTags (char *szData, TAG_TYPE *TagType);

Parameter:

szData:	Receive the Tag ID.
TagType:	Return the tag type.

```
typedef enum TAG_TYPE
{
    Gen1_Tag = 0,
    Gen2_Tag = 1,
    ISO180006B_Tag = 2,
    Unknow_Tag = 10,
}TAG_TYPE;
```

Return code:

TRUE:	Select tag success.
FALSE:	Select tag fail.

8.4 Select Gen2 Tag

Function Description:

To only detect EPC Class1 Gen2 tags.

Function Call:

BOOL SelectGen2Tag (char *szData);

Parameter:

szData: Receive the Tag ID.

Return code:

TRUE: Select tag success.
FALSE: Select tag fail.

8.5 Read Data from Gen2 Tag's Blocks

Function Description:

To read data from Gen2 tag's blocks.

Function Call:

BOOL ReadGen2Block (char *szAddress, int nBlock, char *szData);

Parameter:

szAddress: The start address to read blocks.(For example : To address block #2 of the EPC memory bank, the szAddress is "1002", the "1" specifies the EPC memory bank and "002" specifies the block memory(0x002))
nBlock: The number of blocks which user wants to read.
szData: Receive the block's data.

Return code:

TRUE: Read blocks success.
FALSE: Read blocks fail.

8.6 Write Data to Gen2 Tag's Blocks

Function Description:

To write data to Gen2 tag's blocks.

Function Call:

BOOL WriteGen2Block (char *szAddress, int nBlock, char *szData);

Parameter:

szAddress: The block's address.
nBlock: The number of blocks which user want to write.
szData: The data which user wants to write in the block and 1 block for 2 – bytes data.

Return code:

TRUE: Write block data success.
FALSE: Write block data fail.

8.7 Select ISO18000-6B Tag

Function Description:

To only detect an ISO18000-6B tag.

Function Call:

BOOL SelectISO180006BTags (char *szData);

Parameter:

szData: Receive the Tag ID.

Return code:

TRUE:	Select tag success.
FALSE:	Select tag fail.

8.8 Read Data from ISO18000-6B Tag's Block

Function Description:

To read the data form ISO18000-6B tag's block.

Function Call:

BOOL ReadISO180006BBlock (char *szAddress, int nBlock, char *szTagID, char *szData);

Parameter:

szAddress:	The start address to read blocks.
nBlock:	The number of blocks which user wants to read.
szTagID:	The tag ID which user wants to read.
szData:	Receive the block's data.

Return code:

TRUE:	Read block success.
FALSE:	Read block fail.

8.9 Write Block Data to ISO18000-6B Tag

Function Description:

To write data to ISO18000-6B tag's block .

Function Call:

BOOL WriteISO180006BBlock (char *szAddress, int nBlock, char *szTagID, char *szData);

Parameter:

szAddress:	The start address of the written blocks.
nBlock:	The number of blocks which user want to write.
szTagID:	The tag ID which user wants to write.
szData:	The data which user wants to write in the block and 1 block for 1-byte data.

Return code:

TRUE:	Write block data success.
FALSE:	Write block data fail.

8.10 Select All Tags of Any Type

Function Description:

Selecting all supported tags in field.

Function Call:

BOOL InventoryTag (int *nIndex);

Parameter:

nIndex:	Return the number of tags.

Return code:

TRUE:	Select tag success.
FALSE:	Select tag fail.

8.11 Get The tags from Command InventoryTag

Function Description:

Get the tags which read by command InventoryTag.

Function Call:

void GetTags(TAG_DATA *TagData);

Parameter:

TagData :	The structure of tag information.

```
typedef struct TAG_DATA
{
    char szTag[50];
    TAG_TYPE TagType;
} TAG_DATA;
```

8.12 Send a Tag Password

Function Description:

After assigned an access password to a tag that support passwords, you must send the password to the reader before the reader can execute any other Secure State operations for that tag. This remains a requirement until the password is changed or the password value is reset to zero.

Function Call:

BOOL SendAccessPass (char *szPass);

Parameter:

szPass:	The 4 bytes password.

Return code:

TRUE:	Send password success.
FALSE:	Send password fail.

8.13 Lock Gen2 Tag

Function Description:

Set password protection for the different memory bank of the tag.

Function Call:

BOOL LockGen2Tag (char *szData);

Parameter:

szData:	The lock value.

Return code:

TRUE:	Lock tag success.
FALSE:	Lock tag fail.

8.14 Lock ISO18000-6B Tag

Function Description:

To lock ISO18000-6B blocks. Once the tag blocks have been locked, they cannot be unlocked or written to.

Function Call:

BOOL LockISO180006BTag(char *szTagID, char *szAddress, int nBlock);

Parameter:

szTagID:	The tag ID which user wants to lock.
szAddress:	The start address to lock.
nBlock:	The number of blocks which user wants to lock.

Return code:

TRUE:	Lock tag success.
FALSE:	Lock tag fail.

8.15 Get Reader's Power Level

Function Description:

To get reader's power level.

Function Call:

BOOL GetPowerLevel(int *nPower);

Parameter:

nPower:	Return the power level.

Return code:

TRUE:	Get value success.
FALSE:	Get value fail.

8.16 Set Reader's Power Level

Function Description:

To set reader's power level.

Function Call:

BOOL SetPowerLevel(int nPower, BOOL bSetDefault);

Parameter:

nPower:	The power level.
bSetDefault:	TRUE for set to the value to EEPROM and FALSE for temporary

Return code:

TRUE:	Set value success.
FALSE:	Set value fail.

8.17 Get the library version

Function Description:

To get SkeyDll.dll version.

Function Call:

void GetLibraryVersion(char *szVersion);

Parameter:

szVersion:	Return the version.

8.18 Get Reader's Frequency

Function Description:

To get reader's frequency.

Function Call:

BOOL GetFrequency(FREQUENCY_TYPE nFrequencyType, char *szFrequency);

Parameter:

nFrequencyType: The frequency type.

```
typedef enum FREQUENCY_TYPE
{
    START_FREQUENCY = 1,
    STOP_FREQUENCY = 2,
}FREQUENCY_TYPE;
```

szFrequency: Return the frequency value.

Return code:

TRUE:	Get value success.
FALSE:	Get value fail.

8.19 Set Reader's Frequency

Function Description:

To set reader's frequency.

Function Call:

BOOL SetFrequency(FREQUENCY_TYPE nFrequencyType, char *szFrequency, BOOL bSetDefault);

Parameter:

nFrequencyType:	The frequency type.
szSpacing:	The frequency value.
bSetDefault:	TRUE for set to the value to EEPROM and FALSE for temporary

Return code:

TRUE:	Set value success.
FALSE:	Set value fail.

8.20 Get Reader's Hop Channel Spacing

Function Description:

To get reader's hop channel spacing.

Function Call:

BOOL GetHopChannelSpacing(char *szSpacing);

Parameter:

szSpacing:	Return the hop channel spacing value.

Return code:

TRUE:	Get value success.
FALSE:	Get value fail.

8.21 Set Reader's Hop Channel Spacing

Function Description:

To set reader's hop channel spacing.

Function Call:

BOOL SetHopChannelSpacing(char *szSpacing, BOOL bSetDefault);

Parameter:

szSpacing:	The hop channel spacing.
bSetDefault:	TRUE for set to the value to EEPROM and FALSE for temporary

Return code:

TRUE:	Set value success.
FALSE:	Set value fail.

8.22 Get Reader's Firmware Version

Function Description:

To get reader's firmware version.

Function Call:

BOOL GetFWVersion (char * szVersion);

Parameter:

szVersion:	The reader's firmware version.

Return code:

TRUE:	Get firmware version success.
FALSE:	Get firmware version fail.

8.23 Get Reader's LBT Setting

Function Description:

To get reader's LBT(listen before talk) setting.

Function Call:

BOOL GetListenBeforeTalk (BOOL *bEnable);

Parameter:

bEnable:	The return value to show enable/disable.

Return code:

TRUE:	Get LBT setting success.
FALSE:	Get LBT setting fail.

8.24 Set Reader's LBT Setting

Function Description:

To set reader's LBT(listen before talk) setting.

Function Call:

BOOL SetListenBeforeTalk(BOOL bEnable, BOOL bSetDefault);

Parameter:

bEnable:	The return value to show enable/disable.
bSetDefault:	TRUE for set to the value to EEPROM and FALSE for temporary

Return code:

TRUE:	Set LBT setting success.
FALSE:	Set LBT setting fail.

9 UHF Reader for Kitty

This library “RFID18K6CReader.dll” is used to control the RFID Reader Kitty. Please get it from below URL.
http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip

9.1 Kitty RFID Reader API Reference

To Build the project copy all header file and RFID186KCReader.lib to you project folder, and then include rfidstruct.h and RFID18K6CReaderAPI.h to your project.

Steps of how to use RFID reader interface :
1.Initialize the RFID reader interface
2.Open an RFID reader for control
3.Configure RFID reader
such as Operation mode, response data mode, reader's power state, etc.
4.Configure and enable Antenna
5.Execute Tag Access
such as inventory, read, write, etc.
6.Close the RFID reader
7.Shut down the RFID reader interface.

Note : Before the RFID Reader interface can be used, it must be explicitly initialized. And it must be properly shuts down before an application exits in order to release any internally-held resources.
If an application fails to shut down the RFID Reader interface, the RFID Reader interface will not be available for other applications, it may necessary to reset the RFID Reader module and/or reboot the device.

9.2 Interface Management

9.2.1 Initializing the RFID Reader Interface

Description:
Allows the RFID Reader Interface to properly initialize any internal data structures and put itself into a well-known ready state. This function must be called before any other RFID Reader Interface function.
Function Call:
RFID_STATUS RFIDCreate(char *pszVer);
Parameters:
pszVer - A pointer to a character string, contains the version of the RFID Reader Interface.
Returns: RFID_STATUS.

9.2.2 Shutting Down the RFID Reader Interface

Description:
Allows the RFID Reader Interface to properly clean up any internally-held resources. To prevent resource leaks, an application must ensure that RFID Reader is shut down before the application exits
Function Call:
RFID_STATUS RFIDDestroy();
Returns: RFID_STATUS.

9.3 RFID Reader Configuration

After RFID Reader Interface has been successfully initialized, Developer must open a RFID Reader and then Configure parameters of RFID Reader for the current operation environment. Such as the operation mode, data response format, the power state of RFID Reader and other parameters.

9.3.1 Open RFID Reader

Description:

Open a RFID Reader for control. An application must call this function before control the RFID Reader.

Function Call:

RFID_STATUS RFIDOpen(int nRadio = 0);

Parameters:

nRadio - Reserved for future use.

Returns: RFID_STATUS.

9.3.2 Close RFID Reader

Description:

Release control of a RFID Reader.

Function Call:

RFID_STATUS RFIDClose (int nRadio = 0);

Parameters:

nRadio - Reserved for future use.

Returns: RFID_STATUS.

9.3.3 Set the Operation Mode for the RFID Reader

Description:

The RFID Reader module may operate either in continuous or non-continuous mode. **In continuous mode**, when a tag-protocol-operation cycle (i.e. one iteration through all enabled antenna ports) has completed, the RFID Reader module begins a new tag protocol-operation cycle with the first enabled antenna port and continues to do so until the operation is explicitly cancelled by the application. **In non-continuous mode**, only a single tag-protocol-operation cycle is executed upon the RFID radio module.

Function Call:

RFID_STATUS RFIDSetOperationMode(RFID_RADIO_OPERATION_MODE mode);

Parameters:

mode - The operation mode for the RFID Reader.

Returns: RFID_STATUS.

9.3.4 Get the Operation Mode for the RFID Reader

Description:

Retrieves the operation mode for the RFID Reader.

Function Call:

RFID_RADIO_OPERATION_MODE RFIDGetOperationMode();

Parameters:

None

Returns: RFID_STATUS.

9.3.5 Set the Response Data's Mode of the RFID Reader

Description:
Allows the application to control the mode of data reporting for tag-access operations. By default, the reporting mode is set to "normal". RFID18K6CReaderAPI.dll only support "Compact" and "Normal" format. **Compact mode** contains the minimum amount of data necessary to return the results of tag-protocol operations to the application; **Normal mode** augments compact mode by interleaving additional status/contextual information in the operation results, so that the application can detect. For example, the start of inventory rounds, when a new antenna is being used etc.

Function Call:
RFID_STATUS RFIDSetResponseMode(RFID_RESPONSE_MODE mode);

Parameters:
mode - The requested data-reporting mode. Can be "Normal" or "Compact"

Returns: RFID_STATUS.

9.3.6 Get the Response Data's Mode of the RFID Reader

Description:
Retrieve the mode of data reporting for tag-access operations.

Function Call:
RFID_STATUS RFIDGetResponseMode(RFID_RESPONSE_MODE *pMode);

Parameters:
pMode - A pointer to RFID_RESPONSE_MODE contain the data reporting mode.

Returns: RFID_STATUS.

9.3.7 Set the RFID Reaer's Power State

Description:
Set the RFID Reader module's power state(not to be confused with the antenna RF power).

Function Call:
RFID_STATUS RFIDSetPowerState(RFID_RADIO_POWER_STATE state);

Parameters:
state - the power state for the RFID Reader module

Returns: RFID_STATUS.

9.3.8 Get the RFID Reader's Power State

Description:
Retrieves the radio module's power state(not to be confused with the antenna RF power).

Function Call:
RFID_STATUS RFIDGetPowerState(RFID_RADIO_POWER_STATE *pState);

Parameters:
pState - a pointer to RFID_RADIO_POWER_STATE contain the RFID Reader module's power state.

Returns: RFID_STATUS.

9.3.9 Set the RFID Reader's Low-Level Parameter

Description:
Set the low-level configuration parameter for the RFID Reader module. For example the MAC registry value.

Function Call:
RFID_STATUS RFIDSetConfigurationParameter(INT16U parameter, INT32U value);

Parameters:
parameter – the configuration parameter to set
value – the value to which the configuration parameter will be set.

Returns: RFID_STATUS.

9.3.10 Get the RFID Reader's Low-Level Parameter

Description:
Retrieves a low-level RFID Reader module configuration parameter.
Function Call:
RFID_STATUS RFIDGetConfigurationParameter(INT16U parameter, INT32U *pValue);
Parameters:
parameter – parameter to retrieve.
pValue – pointer to variable that contain the value of configuration parameter.
Returns: RFID_STATUS.

9.4 Antenna Port Configuration

The RFID Reader module supports active use of one or more logical antenna ports, each mapped to a physical transmit and a physical receive antenna port. An application may retrieve status and configure several parameters. These include: Enable/Disable RFID Reader module, Power Level, Dwell Time, Number of Inventory Cycles, Logical-to-Physical Antenna Port Mapping. Please refer to below function description.

9.4.1 Enabling and Disabling Anennas

Description:
Set the state of a RFID Reader module's antenna port.
Function Call:
RFID_STATUS RFIDSetAntennaPortState(INT32 antennaPort,
RFID_ANTENNA_PORT_STATE state);
Parameters:
antennaPort – The logical antenna port to enable or disable.
state – The new state of the logical antenna port.
Returns: RFID_STATUS.

9.4.2 Get the RFID Reader's Antenna-Port Status

Description:
Retrieves the status of a RFID Reader module's antenna port.
Function Call:
RFID_STATUS RFIDGetAntennaPortStatus(INT32U antennaPort,
RFID_ANTENNA_PORT_STATUS *pStatus);
Parameters:
antennaPort – the logical antenna port for which status is to be retrieved.
pStatus – pointer to the structure which will contain the antenna port's status. Must not be NULL.
Returns: RFID_STATUS.

9.4.3 Configuring Antenna-Port Parameters

When configuring or retrieving the configuration for logical antenna port, an application has several parameters that it can set/retrieve. See **RFID_ANTENNA_PORT_CONFIG** structure.
Description:
Allows an application to configure several parameters for a single logical antenna port – for example, dwell time, power level. The application should first retrieve the antenna port's current settings and then, update the values in the structure that are to be changed.
Function Call:
RFID_STATUS RFIDSetAntennaPortConfiguration(INT32U antennaPort, const RFID_ANTENNA_PORT_CONFIG *pConfig);
Parameters:
antennaPort – The logical antenna port to cnfigue.
pConfig – A pointer to the structure that contains the antenna-port configuration parameters. Must not be NULL.
Returns: RFID_STATUS.

9.4.4 Retrieving Antenna-Port Configuration

Description:

Retrieve a single logical antenna port's configurationparameters – for example, dwell time, power level, and number of inventory cycles. Even the logical antenna port is disabled.

Function Call:

RFID_STATUS RFIDGetAntennaPortConfiguration(INT32U antennaPort, RFID_ANTENNA_PORT_CONFIG *pConfig);

Parameters:

antennaPort – The logical antenna port to configure.

pConfig – A pointer to a structure that contains the antenna-port configuration parameter.

Returns: RFID_STATUS.

9.5 ISO 18000-6C Tag Access

The interface support the following Tag access operations:

.Inventory

.Read

.Write

.Kill

.Lock

Tag accesses are comprised of three operations:

Specifying Tag-selection Criteria(pre-singualtion): An application may require that the tag population be logically partitioned into disjoint groups before issuing an access command. After the tags are partitioned, the specified operation may then be applied to one of the groups. An application may specify tag-selection criteria to perform the tag partitioning before tags are singulated.

Apply post-singulation match mask: After the RFID Reader module has singulated a tag, it can optionally apply an application-supplied post-singulation match mask to the singulated tag's EPC to further filter the singulated tag.

Apply ISO 18000-6C access command: Only tags that match the optionally-supplied selection criteria and post-singulation match mask have the access command applied to them.

Note: when perform tag read, write, kill, lock operation, the RFID Reader module uses only the first enabled logical antenna.

Tag Operation Functions

When the application issues a tag operation (i.e. Iventory, read, etc.) to the interface, it also provides a pointer to an application-define callback function. The interface in turn issues the request to the RFID Reader and then returns the operation result via the application-defined callback. Tag operation functions can execute either in Block or Non-block mode. In **Block mode**, the function will waiting for the tag operation to finish and, LPACCESS_STATUS field will contain the operation result. If the timeout lapse this access operation will fail; In **Non-block mod**e, this function will return immediately and the operation result will return by the application-defined callback function.

9.5.1 Callback function

LRESULT (CALLBACK* RFIDPROC)(HWND, UINT, WPARAM, LPARAM);

Description:

Application-define callback function.

Parameters:

HWND – The window issue the tag operation.

UNIT – Undefine reserve for future use

WPARAM – Undefine reserve for future use

LPARAM – a pointer to the structure of ACCESS_STATUS which contain the operation result.

Returns: None

9.5.2 Get Antenna's Response Status

Description:

After the tag access operation executed call this function to retrieve the Antenna's status. In **Block mode** call this function follow the tag access function; In **Non-block mode** call this function in application-defined callback function.

Function Call:

BOOL RFIDGetAntennaStatus(int nAntenna, LPANTENNA_STATUS lpAntennaStatus);

Parameters:

nAntnna – Indicate which antenna's status to be retrieve.

lpAntennaStatus – A pointer to the structure of ANTENNA_STATUS that contain the antenna's status.

Returns:

True/False

9.5.3 Get Tag Access Response Data

Description:

After the tag access operation executed call this function to retrieve the access response data. In **Block mode,** call this function follow the tag access function; In **Non-block mode** call this function in application-defined callback function.

Function Call:

BOOL RFIDGetAccessData(int nAntenna, int nIndex, LPACCESS_DATA lpAccessData);

Parameters:

nAntenna – Indicate which antenna's access data to be retrieve.

nIndex – Index of the access's data

lpAccessData – A pointer to the structure of ACCESS_DATA that contain the access's data.

Returns: True/False

9.5.4 Set Tag Operation Stop Count

Description:

Set the maximum number of tags to which the tag operation will be applied. If this number is zero, then the operation is applied to all selected tags. If this number is non-zero, the antenna-port dwell time and inventory-round-cycles still apply.(For version 1.0, this field may have a maximum value of 1.).

Function Call:

void RFIDSetStopCount(int nStopCount);

Parameters:

nStopCount – the maximum number of tag to which tag operation are applied. Default value is zero.

Returns: None

9.5.5 Get Tag Operation Stop Count

Description:

Get the maximum number of tags to which the tag operation will be applied. For more information see "12.5.4 Set Tag Operation Stop Count".

Function Call:

int RFIDGetStopCount();

Parameters:

None.

Returns:

int – the maximum number of tag.

9.5.6 Tag Inventory Operation

Description:

Executes a tag inventory for all tags of interest. If the selection Criteria and post-singulation is specified, tags will be partitioned first.

Function Call:

void RFIDInventory(RFID_INVENTORY stInventory, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);

Parameters:

stInventory – Inventory operation parameters.
lpAccessStatus - Contain the operation result, use in block mode;
bBlock – Block mode.
nTimeout – Timeout for the access operation, use in block mode.

Returns: None

9.5.7 Tag Read Operation

Description:

Read one or more 16-bit words from any of a tag's memory banks. While a read may be used to retrieve a set of tags EPC data, if the EPC is the only desired data, performing an inventory operation is more efficient. (Read may only be performed on 16-bit word boundaries and for multiples of 16-bit words) if one or more of the memory words specified by the offset/count combinaion do not exist or are read-locked, the read from the tag fails.

Function Call:

void RFIDTagRead(RFID_READ stRead, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagReadEx(RFID_READ_EX stReadEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);

Parameters:

stRead/stReadEx – read operation parameters
lpAccessStatus - Contain the operation result. Use in block mode;
bBlock - Block mode.
nTimeout - Timeout for the access operation, use in block mode.

Returns: None

9.5.8 Tag Write Operation

Description:

Write one or more 16-bit words to he specified memory bank. (Write could only beginning at the specified 16-bit offset. maximum number of 16-bit word to be written is 8).
If wants to write more than eight 16-bit words data to a tag a one time, please use RFIDTagWriteEx.

Function Call:

void RFIDTagWrite(RFID_WRITE stWrite, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagWriteEx(RFID_WRITE_EX stWriteEx, LPACCESS_STATUS lpAccessStatus, int nTimeout = 3000);

Parameters:

stWrite/stWriteEx – write operation parameters
lpAccessStatus - Contain the operation result. Use in block mode;
bBlock - Block mode.
nTimeout - Timeout for the access operation, use in block mode.

Returns: None

9.5.9 Modify EPC Operation

Description:
Modify the target tag's EPC.
Function Call:
void RFIDTagWriteEPC(RFID_WRITE_EPC stEPC, LPACCESS_STATUS lpAccessStatus, int nTimeout = 3000);
Parameters:
stEPC – Contain Modify EPC operation parameters .
lpAccessStatus - Contain the operation result.
nTimeout - Timeout for the access operation.
Returns: None

9.5.10 Tag Kill Operation

Description:
Kill the tags of interest.
Function Call:
void RFIDTagKill(RFID_KILL stKill, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagKillEx(RFID_KILL_EX stKillEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
Parameters:
stKill/stKillEx – Kill operation parameter.
lpAccessStatus - Contain the operation result. Use in block mode;
bBlock - Block mode.
nTimeout - Timeout for the access operation, use in block mode.
Returns: RFID_STATUS

9.5.11 Tag Lock Operation

Description:
Execute a tag lock(setting a tag's access permissions).
Function Call:
void RFIDTagLock(RFID_LOCK stLock, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagLockEx(RFID_LOCK_EX stLockEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock, int nTimeout);
Parameters:
stLock/stLockEx – Lock operation parameter.
lpAccessStatus - Contain the operation result. Use in block mode;
bBlock - Block mode.
nTimeout - Timeout for the access operation, use in block mode.
Returns: RFID_STATUS

9.5.12 Tag Pre-singulation Operation

Description:
Configures the tag-selection criteria for the ISO 18000-6C select command, this command should be issued prior to any tag access operation. The tag-selection criteria will stay in effect until the next call of tag-selection criteria.
Function Call:
RFID_STATUS RFIDTagSelectCriteria(RFID_SELECT_CRITERIA *pCriteria, int *pnCount, BOOL bSet = TRUE);
Parameters:
pCriteria – Contain the tag-selection criteria parameters
pnCount – The number of criteria. This value must be between 0 and 8, inclusive. When set to 0 the tag-selection criteria will be deleted.
bSet – TRUE set the tag-selection criteria parameters;
FALSE retrieve the tag-select criteria parameters.
Returns: RFID_STATUS

9.5.13 Tag Post Singulation Operation

Description:

Configures the post-singulation match criteria to be used by the RFID radio module. An application can use post-singulation to filter tags, based upon all or part of the tag's EPC. The post-singualtion match criteria will stay in effect until the next call of post-singulation.(All the tag's access operation will apply to the which has been singulated only.)

Function Call:

RFID_STATUS RFIDTagPostSingulation(RFID_POST_SINGULATION *pCriteria, int *pnCount, BOOL bSet = TRUE);

Parameters:

pCriteria – Contain post-singulation match criteria parameters.
pnCount – The number of criteria. When set the post-singulation match criteria this value must be 1; When delete the post-singulation match criteria this value must be 0;
bSet – TRUE set the post-singulation parameters;
FALSE retrieve the post-singulation parameters.

Returns: RFID_STATUS

9.5.14 Tag Query Group Operation

Description:

Specifies which tag group will have subsequent access operations applied to it.

Function Call:

RFID_STATUS RFIDTagQueryGroup(RFID_18K6C_TAG_GROUP *pGroup, BOOL bSet = TRUE);

Parameters:

pGroup – Contain the parameters of specifying the tag group.
bSet – TRUE specify the tag group.
FALSE retrieves the tag group.

Returns: RFID_STATUS

9.5.15 Set Current Singulation Algorithm

Description:

Select the current singulation algorithms. Based upon usage scenarios, different singulation algorithms(i.e. Q-adustment) may be desired.

Function Call:

RFID_STATUS RFIDSingulationAlgorithm(
RFID_18K6C_SINGULATION_ALGORITHM *pAlgorithm, BOOL bSet = TRUE);

Parameters:

pAlgorithm – The valid singulation algorityms
bSet – TRUE set the current singulation algorithm
FALSE retrieves the current singulation algorithm.

Returns: RFID_STATUS

9.5.16 Specifying Singulation Algorithm Parameters

Description:

Allow the application to configure the settings for a particular singulation algorithm.

Function Call:

RFID_STATUS RFIDSingulationAlgorithmParameters(
RFID_18K6C_SINGULATION_ALGORITHM algorithm, void *pParms, BOOL bSet = TRUE);

Parameters:

algorithm – The singulation algorithm to be configured.
pParms – A pointer to a structure that contains the singulation algorithm parameters.
bSet – Set or Retrieve the specify singulation alogrithm parameters.

Returns: RFID_STATUS

9.5.17 Cancelling a Tag Operation

Description:
Stops a currently-executing tag operation on a RFID Reader.
Function Call:
RFID_STATUS RFIDCancelOperation ();
Parameters: None
Returns: RFID_STATUS

9.5.18 Aborting a Tag Operation

Description:
Terminate a tag operation immediately. Any response packet will be discarded.
Function Call:
RFID_STATUS RFIDAbortOperation ();
Parameters: None
Returns: RFID_STATUS

9.5.19 Clear RFID Reader Module's Error State

Description:
Clear the error state for the RFID Reader module MAC firmware.
Function Call:
RFID_STATUS RFIDClearError ();
Parameters: None
Returns: RFID_STATUS

9.5.20 Ability of Hold or Discard the Duplicate Tags

Description:
Hold or discard the duplicate tags when inventory. Call this function before inventory.
Function Call:
RFID_STATUS RFIDEnableDuplicate(BOOL bEnable = TRUE);
Parameters:
bEnable – TRUE hold the duplicate tags.
FALSE discard the duplicate tags
Returns: RFID_STATUS

9.6 Other APIs

9.6.1 Get RFID Reader's Firmware Version

Description:
Get RFID Reader's library and Firmware version.
Function Call:
RFID_STATUS RFID MacGetVersion(char *pszVer);
Parameters:
pszVer - Contains the RFID Reader's library and Firmware version.
Returns: RFID_STATUS

9.7 Structure of the Library

9.7.1 RFID_ANTENNA_PORT_CONFIG

Description:

The configuration parameters for a logical antenna port.

Define:

```
typedef struct {
INT32U    length;
INT32U    powerLevel;
INT32U    dwellTime;
INT32U    numberInventoryCycles;
INT32U    physicalRxPort;
INT32U    physicalTxPort;
INT32U    antennaSenseThreshold;
} RFID_ANTENNA_PORT_CONFIG;
```

Fields:

length - The length of the structure in bytes. Must be set to by the application to sizeof(RFID_ANTENNA_PORT_CONFIG)

powerLevel - The power level for the logical antenna port's physical transmit antenna. This value is specified in 0.1 (i.e., 1/10th) dBm. the value must between 0 and 300 and, 280 is The most appropriate value.

dwellTime - The number of milliseconds to spend on this antenna port during a cycle. Zero indicates that antenna usage will be ontrolled by the numberInventoryCycles field.

numberInventoryCycles - The number of inventory rounds to perform with this antenna port. Zero indicates that the antenna usage will be controlled by the dwellTime field.

physicalRxPort - he underlying physical receive antenna port associated with the logical antenna port. Must be between 0 and 3, inclusive.

physicalTxPort - The underlying physical transmit antenna port associated with the logical antenna port. Must be between 0 and 3, inclusive.

antennaSenseThreshold - The measured resistance, specified in ohms.

Note: *In version 1.0 physicalRxPort and physicalTxPort's value must be the same. *dwellTime and numberInventoryCycles can not both be zero. *length field must be fill with the length of the structure before call the config function.

9.7.2 ACCESS_STATUS

Description:

Contain the information of the tag operation result.

Define:

```
typedef struct ACCESS_STATUS_TAG{
FILETIME ftStartTime;
FILETIME ftEndTime;
INT32U    dwResponseMode;
INT32U    dwOperationMode;
INT16U    unCommand;
DWORD     dwErrorCode;
DWORD     dwStatus;
INT16U     unAntennas
} ACCESS_STATUS, *LPACCESS_STATUS;
```

Fields:

ftStartTime – Time of the Tag operation start

ftEndTime – Time of the Tag operation end

dwResponseMode – Operation Response data reporting mode can be Compat or Normal.

dwOperationMode – Working mode of the RFID Reader module can be continuous or non-continuous.

dwErrorCode – If tag operation failed this field contains the error code. Zero indicates no error.

dwStatus – Return by the Tag operation function indicate the library status and error codes. Zero indicates no error.

unAntennas – The amount of antennas.

9.7.3 ANTENNA_STATUS

Description:

Contains the information of the antenna that take effect in the tag operation executed.

Define:

```
typedef struct ANTENNA_STATUS_TAG{
FILETIME ftStartTime;
FILETIME ftEndTime;
INT16U    unAntenna;
DWORD     dwErrorCode;
DWORD     dwStatus;
INT16U     unCount
} ANTENNA_STATUS, *LPANTENNA_STATUS;
```

Fields:

ftStartTime – Time of this antenna start to execute Tag operation.

ftEndTime – Time of this antenna finish execute Tag operation.

unAntenna – Antenna No.

unCount – The number of tag this antenna returns.

9.7.4 ACCESS_DATA

Description:

Contain access data of the tag return by the antenna.

Define:

```
typedef struct ACCESS_ DATA _TAG{
INT16U unEPCLength;
INT16U unRSSI;    //The receive signal strength
BYTE pnEPC[68]; //PC + EPC + CRC
INT16U    unDataLength;
BYTE pnData[256];                    //Access data or access Status
} ACCESS_DATA, *LPACCESS_DATA;
```

Fields:

unEPCLength–The length of the EPC(include PC and CRC)
unRSSI – The receive signal strength.
pnEPC[68] – Data of EPC(2 bytes PC + EPC + 2 bytes CRC).
unDataLength – Access Data length.
pnData[256] – Data retrieve from Tag of write to Tag.

9.7.5 RFID_INVENTORY

Description:

Inventory operation parameters.

Define:

```
typedef struct RFID_INVENTORY_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
}RFID_INVENTORY;
```

Fields:

hWnd – The parent window which issue the Tag operation.
lpfnStartProc – Application-defined callback function. Thre RFID Reader Interface call this function before execute Tag operation.
lpfnStopProc – Application-defined callback function. Thre RFID Reader Interface call this function after finished Tag operation.

9.7.6 RFID_READ

Description:

Tag Read Operation parameters.

Define:

```
typedef struct RFID_READ_TAG {
HWND hWnd;      //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
RFID_18K6C_MEMORY_BANK bank;
INT16U offset;
INT16U count;
INT32U accessPassword;
}RFID_READ;
```

Fields:

hWnd - The Parent window which issue the Tag operation.

lpfnStartProc - Application-defined callback function. The RFID Reader Interface call this function before execute Tag operation.

lpfnStopProc - Application-defined callback function. The RFID Reader Interface call this function after finished Tag operation.

bank – The memory bank from which to read.

offset – The offset of the first 16-bit word to read from the specified memory bank.

count – The number of 16-bit words to read.(If this value zero and bank is EPC, the read returns the contents of the EPC starting at the 16-bit word specified by offset through the end of the EPC.This value must be in the range 1 to 255, inclusive.)

accessPassword – The access passowrd for the tags. A value of zero indicates no access password.

9.7.7 RFID_READ_EX

Description:

Tag read extension function's parameter.

Define:

```
typedef struct RFID_READ_EX_TAG {
HWND hWnd;      //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
RFID_18K6C_MEMORY_BANK bank;
INT16U offset;
INT16U count;
BYTE accessPassword[8];
}RFID_READ_EX;
```

Fields:

hWnd - the same as RFID_READ.

lpfnStartProc – the same as RFID_READ;

lpfnStopProc – the same as RFID_READ;

bank – the same as RFID_READ;

offset – the same as RFID_READ;

count – the same as RFID_READ;

accessPassword[8] – The access passowrd for the tags. It's 8 bytes' hex char(0~f).

9.7.8 RFID_WRITE

Description:

Tag write operation's parameter.

Define:

```
typedef struct RFID_WRITE_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
BOOL32 verify;
INT32U verifyRetryCount;
RFID_18K6C_MEMORY_BANK bank;
INT32U accessPassword;
INT16U offset;
INT16U count;
Int16U pnData[8];
}RFID_WRITE;
```

Fields:

hWnd – The parent window which issue the Tag operation.

lpfnStartProc - Application-defined callback function. The RFID Reader Interface call this function before execute Tag operation.

lpfnStopProc - Application-defined callback function. The RFID Reader Interface call this function after finished Tag operation.

verify – A flag that indicates if the data written to the tag should be read back from the tag to verify that it was successfully written. A non-zero value indicates that the tag's memory should be read to verify.

verifyRetryCount – The maximum number of times the write should be retried if the write-verify failure. This value must be between 0 and 7.

accessPassword – The access password for the tags. A value of zero indicates no access password.

bank – The memory bank from which to read.

offset – The offset of the first 16-bit word to read from the specified memory bank.

count – The number of 16-bit words to read.(If this value zero and bank is EPC, the read returns the contents of the EPC starting at the 16-bit word specified by offset through the end of the EPC.This value must be in the range 1 to 8, inclusive.)

pnData[8] – Contains the data to be written to the tag's specified memory bank. The high-order byte of pnData[n] is written to the tag's memory-bank byte at 16-bit offset(offset + n). The low-order byte is write to the next byte.

9.7.9 RFID_WRITE_EX

Description:

Tag write extension function's parameter.

Define:

```
typedef struct RFID_WRITE_EX_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
BOOL32 verify;
INT32U verifyRetryCount;
BYTE accessPassword[8];          //password is 8 bytes' hex char(0~f)
BYTE pnEPC[64];
RFID_18K6C_MEMORY_BANK bank;
INT16U offset;
INT16U count;
Int16U pnData[256];          //data to be writen(hex chars)
}RFID_WRITE_EX;
```

Fields:

hWnd – the same as RFID_WRITE

lpfnStartProc - the same as RFID_WRITE

lpfnStopProc - the same as RFID_WRITE

verify – the same as RFID_WRITE

verifyRetryCount – the same as RFID_WRITE

accessPassword[8] – The access passowrd for the tags. It's 8 bytes' hex char(0~f)

pnEPC[64] – EPC of the specify tag to be writed to. If this parameter is NULL, the data will be written to the tags in the field. Otherwise data will be written to the tag specify by the EPC.

bank – the same as RFID_WRITE

offset – the same as RFID_WRITE

count – the same as RFID_WRITE

pnData[256] – Contains the data to be written to the tag's specified memory bank(HEX chars' string).

9.7.10 RFID_WRITE_EPC

Description:

The structure of Modify a tag's EPC.

Define:

```
typedef struct RFID_WRITE_EPC_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
BOOL32 verify;
INT32U verifyRetryCount;
BYTE accessPassword[8];
BYTE pnOldEPC[64];
INT16U tagType;
BYTE pnNewEPC[64];
}RFID_WRITE_EPC;
```

Fields:

hWnd - the same as RFID_WRITE.

lpfnStartProc – the same as RFID_WRITE

lpfnStopProc – the same as RFID_WRITE

verify – the same as RFID_WRITE

verifyRetryCount – the same as RFID_WRITE

accessPassword[8] – the same as RFID_WRITE_EX

pnOldEPC[64] – the old EPC of the tag to be modify

tagType – Reserve for future use.

PnNewEPC[64] – the new EPC of the target tag.

9.7.11 RFID_KILL

Description:

The structure of Kill a tag.

Define:

```
typedef struct RFID_KILL_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
INT32U accessPassword;
INT32U killPassword;
}RFID_KILL;
```

Fields:

hWnd –The Parent window which issue the Tag operation.

lpfnStartProc – Application-defined callback function. The RFID Reader Interface call this function before execute Tag operation.

lpfnStopProc – Application-defined callback function. The RFID Reader Interface call this function after finished Tag operation.

accessPassword – The access password for the tags. A valueof zero indicates no access password.

killPassword – The kill password for the tags. Must not be zero.

9.7.12 RFID_KILL_EX

Description:

Tag Kill extension function's parameter.

Define:

```
typedef struct RFID_KILL_EX_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
BYTE accessPassword[8]; //password is 8 bytes' hex char(0~f)
BYTE killPassword[8];     //password is 8 bytes' hex char(0~f)
}RFID_KILL_EX;
```

Fields:

hWnd - the same as RFID_ KILL.

lpfnStartProc – the same as RFID_KILL.

lpfnStopProc – the same as RFID_KILL.

accessPassword[8] – The access passowrd for the tags. It’s 8 bytes' hex char(0~f).

killPassword[8] – The kill passowrd for the tags. It’s 8bytes' hex char(0~f).

9.7.13 RFID_LOCK

Description:

The structure of Tag locks function.

Define:

```
typedef struct RFID_LOCK_TAG {
HWND hWnd;      //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
INT32U accessPassword;
INT32U killPasswordPermissions;
INT32U accessPasswordPermissions;
INT32U epcBankPermissions;
INT32U tidBankPermissions;
INT32U userBankPermissions;
}RFID_LOCK;
```

Fields:

hWnd –The Parent window which issue the Tag operation.
lpfnStartProc – Application-defined callback function. The RFID Reader Interface call this function before execute Tag operation.
lpfnStopProc – Application-defined callback function. The RFID Reader Interface call this function after finished Tag operation.
accessPassword - Access password for the tags(zero indicates no access password).
killPasswordPermissions - The access permissions for the tag's kill password.
accessPasswordPermissions - The access permissions for the tag's access password.
epcBankPermissions - The access permissions for the tag's EPC memory bank.
tidBankPermissions - The access permissions for the tag's TID memory bank.
userBankPermissions - The access permissions for the tag's USER memory bank.

The access permissions can be set as below:

Name	Value	Description
Allow	0	Password permission: Can be read and write. Bank permission: The memory bank is writeable.
Always Allow	1	Password permission: Can be read and write. Bank permission: The memory bank is writeable. This permission is set permanently.
Password protected	2	Password permission: Need password to read and write. Bank permission: Need password to write to the memory bank.
Always Deny	3	Password permission: Can not be read and write. When the kill password permission is set to be this value, the tag can not be killed. Bank permission: The memory bank can not be written. This permission is set permanently.
No Change	4	The permission should remain unchanged.

Access permissions' values' Table

9.7.14 RFID_LOCK_EX

Description:

Tag Lock extension function's parameter.

Define:

```
typedef struct RFID_LOCK_EX_TAG {
HWND hWnd;       //Parent window
RFIDPROC lpfnStartProc;
RFIDPROC lpfnStopProc;
BYTE accessPassword[8];
INT32U killPasswordPermissions;
INT32U accessPasswordPermissions;
INT32U epcBankPermissions;
INT32U tidBankPermissions;
INT32U userBankPermissions;
}RFID_LOCK_EX;
```

Fields:

hWnd –The same as RFID_LOCK.
lpfnStartProc –The same as RFID_LOCK.
lpfnStopProc –The same as RFID_LOCK..
accessPassword[8] - The access passowrd for the tags. It's 8 bytes' hex char(0~f).
killPasswordPermissions - The same as RFID_LOCK.
accessPasswordPermissions - The same as RFID_LOCK.
epcBankPermissions - The same as RFID_LOCK.
tidBankPermissions - The same as RFID_LOCK.
userBankPermissions - The same as RFID_LOCK.

9.7.15 RFID_SELECT_CRITERIA

Description:

The structure of pre-singulation matches criteria.

Define:

```
typedef struct RFID_SELECT_CRITERIA_TAG {
RFID_18K6C_MEMORY_BANK bank;
INT32U offset;
INT32U count;
INT8U mask[RFID_18K6C_SELECT_MASKBYTE_LEN];
RFID_18K6C_TARGET target;
RFID_18K6C_ACTION action;
BOOL32 enableTruncate;
 }RFID_SELECT_CRITERIA
```

Fields:

bank - The memory bank to match against
offset - The offset of the first bit to match
count - The number of bits in the mask
mask[RFID_18K6C_SELECT_MASK_BYTE_LEN] - The bit pattern to match.
target - What will be affected by the action(S0~S4,SL)
action - The action which will be performed upon the tag populations (i.e, matching and non-matching) during the selection.
enableTruncate – Should the EPC be truncated when the tag is singulated? Non-zero value indicate that the EPC is truncated. If enableTruncate is true: bank must be EPC; Target must be RFID_18K6C_TARGET_SELECTED_FLAG

9.7.16 RFID_POST_SINGULATION

Description:

The structure of post-singulation matches criteria.

Define:

```
typedef struct RFID_POST_SINGULATION_TAG {
INT32U offset;
INT32U count;
INT8U mask[RFID_18K6C_SINGULATION_MASK_BYTE_LEN];
BOOL32 match;
}RFID_POST_SINGULATION;
```

Fields:

offset – offset in bits, from the start of the EPC.

count –The number of bits in the mask.Alength of zero causes all EPC to match. If (offset + count) falls beyond the end of the mask, the tag is considered non-matching.(valid values are 0 to 396).

mask[RFID_18K6C_SINGULATION_MASK_BYTE_LEN] – the bit pattern to match.(hex chars).

match – Determines if the associated tag operation will be applied to tags that match the maskor not.

9.8 Error Code

There are two types of error codes. One is returned by RFID Library interface indicates RFID library status and the other is return by the RFID module's firmware contains access operation result.

RFID Library status and error codes:

Name	*Value*	
RFID_STATUS_OK	0	Success
RFID_ERROR_ALREADY_OPEN	-9999	Attempted to open a radio that is already open
RFID_ERROR_BUFFER_TOO_SMALL	-9998	Buffer supplied is too small
RFID_ERROR_FAILURE	-9997	General failure
RFID_ERROR_DRIVER_LOAD	-9996	Failed to load radio bus driver
RFID_ERROR_DRIVER_MISMATCH	-9995	Library cannot use version of radio bus driver present on system
RFID_ERROR_EMULATION_MODE	-9994	Operation cannot be performed while library is in emulation mode
RFID_ERROR_INVALID_ANTENNA	-9993	Antenna number is invalid
RFID_ERROR_INVALID_HANDLE	-9992	Radio handle provided is invalid
RFID_ERROR_INVALID_PARAMETER	-9991	One of the parameters to the function is invalid
RFID_ERROR_NO_SUCH_RADIO	-9990	Attempted to open a non-existent radio
RFID_ERROR_NOT_INITIALIZED	-9989	Library has not been successfully initialized
RFID_ERROR_NOT_SUPPORTED	-9988	Function not supported
RFID_ERROR_OPERATION_CANCELLED	-9987	Operation was cancelled by call to cancel operation, close radio, or shut down the library
RFID_ERROR_OUT_OF_MEMORY	-9986	Library encountered an error allocating memory
RFID_ERROR_RADIO_BUSY	-9985	The operation cannot be performed because the radio is currently busy
RFID_ERROR_RADIO_FAILURE	-9984	The underlying radio module encountered an error
RFID_ERROR_RADIO_NOT_PRESENT	-9983	The radio has been detached from the system
RFID_ERROR_CURRENTLY_NOT_ALLOWED	-9982	The RFID library function is not allowed at this time
RFID_ERROR_RADIO_NOT_RESPONDING	-9981	The radio module's MAC firmware is not responding to requests
RFID_ERROR_NONVOLATILE_INIT_FAILED	-9980	The MAC firmware encountered an error while initiating the nonvolatile memory update. The MAC firmware will return to its normal idle state without resetting the radio module
RFID_ERROR_NONVOLATILE_OUT_OF_BOUNDS	-9979	An attempt was made to write data to an address that is not in the valid range of radio module nonvolatile memory addresses.
RFID_ERROR_NONVOLATILE_WRITE_FAILED	-9978	The MAC firmware encountered an error while trying to write to the radio module's nonvolatile memory region.
RFID_ERROR_RECEIVE_OVERFLOW	-9977	The underlying transport layer detected that there was an overflow error resulting in one or more bytes of the incoming data being dropped. The operation was aborted and all data in the pipeline was flushed.

Access Operation Result:

Value	*Description*
0x00	Success
0x01	Read after write verify failed.
0x02	problem transmitting tag command
0x03	CRC error on tag response to a write
0x04	CRC error on the read packet when verifying the write
0x05	Maximum retry's on the write exceeded
0x06	Failed waiting for read data from tag, possible timeout.
0x07	Failure requesting a new tag handle.
0x0A	error waiting for tag response, possible timeout
0x0B	CRC error on tag response to a kill
0x0C	Problem transmitting 2nd half of tag kill.
0x0D	tag responded with an invalid handle on first kill command
0xFA	tag has insufficient power to perform the memory write
0xFB	specified memory location is locked and/or permalocked
0xFC	specified memory location does not exist
0xFD	Tag failed to response within timeout
0xFE	CRC was invalid
0xFF	general error

9.9 Support Dot Net Compact Framework

R1000ReaderCF.dll is a C# wrap of Native C/C++ DLL of RFID18k6cReader.dll. It provides APIs for Dot Net Compact Framework programmer to control the RFID Reader.

9.9.1 Class "R1000Reader"

This is main class instantiated by applications. It provides properties and methods for accessing the RFID Reader. The methods in R1000ReaderCF have the same name as their Native implementation.

9.9.2 Programming Model

The first is add reference of R1000ReaderCF.dll to your project.

C# example:

```
using Unitech.R1000.Reader;
using Unitech.R1000.Reader.Constants;
using Unitech.R1000.Reader.Structures;

//Create and Initialize the RFID Library
String strVersion = String.Empty;
R1000Reader.RFIDCreate(ref strVersion);
//Open the RFID Reader
R1000Reader.RFIDOpen(0);
//Inventory & retrieve the tag's epc
RFID_INVENTORY stInventory = new RFID_INVENTORY();
ACCESS_STATUS stAccessStatus = new ACCESS_STATUS();
//operation in blocking mode
R1000Reader.RFIDInventory(stInventory, ref stAccessStatus, true, 3000);
if (stAccessStatus .dwStatus == 0 && stAccessStatus.dwErrorCode == 0)
{
    for (int i = 0; i < stAccessStatus.unAntennas; i++)
    {
        ANTENNA_STATUS stAntennaStatus = new ANTENNA_STATUS();
        R1000Reader.RFIDGetAntennaStatus(i, ref stAntennaStatus);
        for (int j = 0; j < stAntennaStatus.unCount; j++)
        {
            ACCESS_DATA accessData = new ACCESS_DATA();
            UInt32 nRet = R1000Reader.RFIDGetAccessData(i, j, ref accessData);
            if (nRet == 1 && accessData.unEPCLength > 0)
            {
                //Get EPC from accessData.pnEPC;
            }
        }
    }
}
//Close the RFID Reader
R1000Reader.RFIDClose(0);
//Destroy RFID Library
R1000Reader.RFIDDestroy();
```

10 Useful function call – without include SysIOAPI.DLL

Below API maybe useful for you to control HT6xx/PA96x

10.1.1 Warm-boot. Cold-boot and power off

```
#include    <pkfuncs.h>
#include    "oemioctl.h"

//   Warn boot
KernelIoControl(IOCTL_HAL_REBOOT, NULL, 0, NULL, 0, NULL);

// Cold boot
KernelIoControl(IOCTL_COLD_BOOT, NULL, 0, NULL, 0, NULL);

// Power off
{
      DWORD dwExtraInfo=0;
      BYTE bScan=0;
      keybd_event( VK_OFF, bScan, KEYEVENTF_SILENT, dwExtraInfo );
      keybd_event( VK_OFF, bScan, KEYEVENTF_KEYUP, dwExtraInfo );
}
```

11 Get Device ID

In unitech device, an unique ID had been burnt into terminal, user can check it by pressing “Func”+”9”.

The sample code for read device ID as follow,

```
//////////////////////////////////////////////////////////////
HWND hDeviceId = GetDlgItem(hWnd, IDC_DEVICEID);

PDEVICE_ID pDeviceID = NULL;
TCHAR outBuf[512], deviceID[200];
DWORD bytesReturned;
char platformID[64];

pDeviceID = (PDEVICE_ID)outBuf;
pDeviceID->dwSize = sizeof(outBuf);
if (KernelIoControl(IOCTL_HAL_GET_DEVICEID, NULL, 0, outBuf, sizeof(outBuf), &bytesReturned))
{
            // Platform ID
            memcpy((PBYTE)platformID, (PBYTE)pDeviceID + pDeviceID->dwPlatformIDOffset, pDeviceID->dwPlatformIDBytes);

            // Device ID for WinCE version
            memcpy((PBYTE)deviceID, (PBYTE)pDeviceID + pDeviceID->dwPresetIDOffset, pDeviceID->dwPresetIDBytes);
            swprintf(szProductID, _T("%s"), stringBuffer);


            // Device ID for Mobile version
            memcpy((PBYTE) szBuff, (PBYTE)pDeviceID + pDeviceID->dwPresetIDOffset, pDeviceID->dwPresetIDBytes);
            swprintf(deviceID, TEXT("%X%X%X%X%X%X%X%X"),
                        szBuff [0], szBuff [1], szBuff [2], szBuff [3], szBuff [4], szBuff [5], szBuff [6], szBuff [7]);

}
//////////////////////////////////////////////////////
```

The code will have platformID holds Platform ID, and deviceID holds Device ID.

12 Get OEM Info

In HT6xx/PA96x, an OEM ID had been burnt into terminal, user can check it by pressing “Func”+”9”.

The sample code for read OEM ID as follow,

```
////////////////////////////////////////////////////////
{
        TCHAR szBuff[500];

        ZeroMemory(szBuff, sizeof(szBuff));

        SystemParametersInfo(SPI_GETOEMINFO, 500, (LPVOID)szBuff, 0);

        MessageBox(szBuff);
}
////////////////////////////////////////////////////////
```

13 Get firmware and bootloader version info

```
////////////////////////////////////////////////////////
{
        DEVICE_CONFIG cfg;
        if (KernelIoControl(IOCTL_GET_DEVICE_CONFIG, NULL, 0, &cfg, sizeof(cfg), NULL))
        {
                //Get OS Version
                mbstowcs(stringBuffer, cfg.swVersion, strlen(cfg.swVersion));
                swprintf(szProductID, _T("OS version : %s"), stringBuffer);
                Show(szProductID);

                //Get Bootloader Version
                memset (stringBuffer, 0, sizeof(stringBuffer));
                mbstowcs(stringBuffer, cfg.bootloadversion, strlen(cfg.bootloadversion));
                wsprintf(szProductID, _T("Bootloader version : %s"), stringBuffer);
                Show(szProductID);
        }
}
////////////////////////////////////////////////////////
```

14 Camera SDK

Please get SDK from below URL.
http://w3.tw.ute.com/pub/cs/SDK/Camera/CameraSDK.zip

Note: This SDK only for PA550 and PA690.

15 Fingerprint related functions

Please get sample program and manual from below URL.
http://w3.tw.ute.com/pub/cs/software/Sample_Program/PA968/Fingerprint.zip

Note: Only for PA968

16 *GPS related functions*

Please get sample program and manual from below URL(Only for WinCE Version).
http://w3.tw.ute.com/pub/cs/software/Sample_Program/GPS/GPSSDK.zip

17 *USI .NET Compact Framework Component*

Please get sample program and manual from below URL.
http://w3.tw.ute.com/pub/cs/SDK/USI/USICF.zip

18 USI ActiveX Control

Please get binary file and html example from below URL.
http://w3.tw.ute.com/pub/cs/software/Sample_Program/USIActiveX/USIActiveX.zip

18.1 Register Control

a. Copy Microsoft “REGSVRCE.exe” to device.
b. Run “REGSVRCE.exe ScannerActiveX.dll” to register control.
c. Warmboot device to apply system change.

18.2 Embedded to html

```
<OBJECT ID="Scanner"
    CLASSID="CLSID:E81DD955-9B99-4493-8035-355DFB5028D9"
    WIDTH=0 HEIGHT=0>
</OBJECT>
```

18.3 *Operate control by script language*

a. Enable Scanner:

```
<SCRIPT LANGUAGE="Javascript">
      function OnRegister()     {     Scanner.Register=1     }
      function OnUnregister()   {     Scanner.Register=0     }
      function OnEnable()       {     Scanner.Scan=1         }
      function OnDisable()      {     Scanner.Scan=0         }
</SCRIPT>

<INPUT NAME="REGISTER1" TYPE="BUTTON" VALUE="Register" onClick="OnRegister()" >
<INPUT NAME="ENABLE" TYPE="BUTTON" VALUE="Enable" onClick="OnEnable()" >
<INPUT NAME="DISABLE1" TYPE="BUTTON" VALUE="Disable" onClick="OnDisable()" >
<INPUT NAME="UNREGISTER" TYPE="BUTTON" VALUE="Unregister" onClick="OnUnregister()" >
```

b. Change Hamster Setting:

```
<SCRIPT LANGUAGE="Javascript">
        function OnUPCEnable()
        {
                Scanner.SetHamster(0x79,1);
          }

        function OnUPCDisable()
        {
                Scanner.SetHamster(0x79,0);
          }
</SCRIPT>

<INPUT NAME="UPC_Enable" TYPE="BUTTON" VALUE="UPC E Enable" onClick="OnUPCEnable()" >
<INPUT NAME="UPC_Disable" TYPE="BUTTON" VALUE="UPC E Disable" onClick="OnUPCDisable()" >
```

19 32WAN GPRS library

Please get sample program and manual from below URL(Only for WinCE Version).
http://w3.tw.ute.com/pub/cs/software/Sample_Program/32WAN/32WAN_SDK.zip

20 Update notes

V1.0 The first version
V1.1 Wrong URL link for C# on chapter 1.4
V1.2 PA982 support
V1.3 Add RH767 HF/UHF programming on chapter 9 & 10
V1.4 Modify RH767 HF programming on chapter 9.
V1.6 Change logo
V1.7 Modify SDK URL
V1.8 Add RH767 UHF SkyeTek programming on chapter 11
V1.9 Add HF API in chapter 9.
V1.10 PA968 support and include camera, fingerprint, GPS and GPRS programming guide
V1.11 Update the HF API.
V1.12 Remove HF multi tag API.
V1.13 Add Matrix 2 of 5 supporting which sharing setting with Toshiba code on page 14, 15 and 22
V1.14 Add description about retrieve firmware and bootloader version info.
V1.15 Add RFID reader Kitty
V1.16 Modify for HT680.
V1.17 Document error - SD/Vibration API for PA968
Modify RFID SDK link
V1.18 Modify the HF reader error code table
V1.19 Modify for the device with Windows Mobile system.
V1.20 Modify for PA690
V1.21 Modify GPS link.
Modify RFID SDK link.
Remove Chapter Scanner3, ScanKey3, BTAPI
Modify Chapter about Camera