unitech

MR650 Programming Manual

V 1.16 11/27/2008

1. INTRODUCTION 9
1.1. Why SDK? 9
1.2. How to download data from building RFID proximity reader 9
1.3. COM definition for MR650 9
1.4. Useful Sample program 10
2. USI.DLL - UNITECH SCANNER INTERFACE DLL 11
2.1. Register the application to the USI DLL 11
2.2. Unregister the application from the USI.DLL 12
2.3. Enable / Disable Scanner 12
2.4. Reset Scanner 12
2.5. Get error code 12
2.6. Return the system error code 12
2.7. Get scan data 13
2.8. Get length of scanned data 14
2.9. Get Symbology name 14
2.10. Clear scan data system buffer 15
2.11. Good read indicator 15
2.12. Wait for acknowledgement of the last sent command 15
2.13. Save setting to profiles 15
2.14. Save scanner setting into specified file 15
2.15. Change scanner setting from specified setting profile 16
2.16. Automatically enable scanner beam with pressing trigger key 16
2.17. Stop auto scanning function 16
2.18. Check if auto scanning is enable 16
2.19. Check if Scan2Key.exe program is running or not 16
2.20. Test if Scan2Key is enabled 18
2.21. Load/Unload Scan2Key.exe 18

2.22. Enable/Disable Scan2Key 18

2.23. Send scanner command to decoding chip 18

2.24. Only send single command decoding chip 19

2.25. Send command to decoding chip 19

3. CONTROL COMMAND FOR DECODER CHIP 20

4. UNITECHAPI.DLL 25

4.1. Disable ActiveSync 25

4.2. Enable ActiveSync 25

4.3. Suspend 25

4.4. Disable TaskBar 25

4.5. Enable TaskBar 26

4.6. Disable Desktop 26

4.7. Enable Desktop 26

4.8. Disable toolbar on windows explorer 26

4.9. Enable toolbar on windows explorer 26

5. SYSIOAPI.DLL (FOR HARDWARE RELATED IO CONTROL API) 27

5.1. IO Device Control Method 28
5.1.1. Create IO Device 28
5.1.2. Close Device 28

5.2. Access Relays 29
5.2.1. Check Relay Status 29
5.2.2. Set Relay 30

5.3. Access Optical Isolated Input 31
5.3.1. Get Optical Isolated Input Status 31
5.3.2. Event for Optical Input Change 32

5.4. Get Back Cover Open/Close status 33

5.5. Power Control 35
5.5.1. Get Module 1 Power Status 35
5.5.2. Control Module 1 Power 36

5.6. PCMCIA Control 37
5.6.1. Check CF Slot 37
5.6.2. Enable/Disable CF Slot 38

5.7. Camera Control 39

5.7.1. Power on/off Camera 39
5.7.2. Camera Reset 39

5.8. Watch Dog 40
5.8.1. Set time out value 40
5.8.2. Start the watchdog 40
5.8.3. Pause the watchdog 40

6. CAMERA CONTROL - CAMERADLL.DLL 41

6.1. Open Camera 41

6.2. Close Camera 41

6.3. Preview CCD video input 41

6.4. Stop Preview CCD video input 41

6.5. Capture image 41

6.6. Get color key 42

6.7. Set color key 42

6.8. Start video capture 42

6.9. End video capture 42

6.10. Video playback function 43
6.10.1. Initiate playback 43
6.10.2. Release playback 43
6.10.3. Start playback 43
6.10.4. Stop playback 43
6.10.5. Pause playback 43
6.10.6. Continue playback 43
6.10.7. Check if playback 44

6.11. Convert raw MPEG to standard MPEG 44

7. FINGER PRINT CONTROL BIOIDDLL.DLL 45

7.1. Power On/Off the Finger Print Module 45

7.2. Initialize Finger Print Library 45

7.3. Uninitialize Finger Print Library 45

7.4. Connect to Finger Print Module 45

7.5. Disconnect to Finger Print Module 46

7.6. GUI Callback Function 46

7.7. Release Memory Block 47

7.8. Get the Image of Living Finger 47

7.9. Get the Image of Living Finger by Defined Window 48
7.10. Save Living Finger Print Template to Module 49
7.11. Finger Print Verify 50
7.12. Finger Print Verify Ex 51
7.13. Verify Finger Print with All Templates in the Module 52
7.14. Detect Finger Print 53
7.15. Store Finger Print Tamelate to Module 53
7.16. Delete the Finger Print in the Module 53
7.17. Remove All Finger Print in Module 54
7.18. Store Finger Print to Module 54
7.19. Read Finger Print From Module 54
7.20. List All Finger Print in the Module 55
7.21. Get the Finger Print Template from Specific Slot 55
7.22. Get Module Information 55
7.23. Set the Session Parameter of Module 56
7.24. Get the Session Parameter of Module 56
7.25. Get Remaining EEPROM Memory in the Module 56
7.26. Get Living Finger Print Template 57

8. USEFUL FUNCTION CALL - WITHOUT INCLUDE SYSIOAPI.DLL 58

8.1. Warm-boot, Cold-boot and power off 58

9. MIFARE READER LIBRARY 59

9.1. Connect to Mifare Reader 59
9.2. Disconnect with Mifare Reader 59
9.3. Check the connection with Mifare Reader 59
9.4. Firmware Version 59
9.5. Write Key to EEPROM 60
9.6. Load Key from EEPROM 60
9.7. Load Key with User Defined 60

9.8. Read Card's Block Data 60
9.9. Write Card's Block Data 61
9.10. Read Card's Value 61
9.11. Write Card's Value 61
9.12. Increase Value 61
9.13. Decrease Value 62
9.14. Read Sector Data 62
9.15. Read Multi Sectors Data 62
9.16. Reader Serial Number 62
9.17. Start Read Card Serial Number - I 63
9.18. Start Read Card Serial Number - II 63
9.19. Stop Read Card Serial Number 63
9.20. Get Card Serial Number 63
9.21. Check Reading 63
9.22. Get Library Version 64
9.23. Error Code 64
10. FINGER PRINT CONTROL BIOIDDLL.DLL(BIOSCRYPT) 65
10.1. Start Finger print function 65
10.2. Stop Finger print function 65
10.3. Connect to Finger print module 65
10.4. Disconnect Finger print module 65
10.5. Setup Finger print module communication type 66
10.6. Get module's communication type 66
10.7. Set module's communication type 67
10.8. Get FP module's baudrate 67
10.9. Set FP module's baudrate 67
10.10. Get FP module's Aux Port baudrate 68
10.11. Set FP module's Aux Port baudrate 68

10.12. Set FP module's Aux Port baudrate 68
10.13. Get FP version 68
10.14. Initialize FP directory 69
10.15. Read FP directory 69
10.16. Get template from FP module 69
10.17. Send template to FP module 70
10.18. Remove template from FP module 70
10.19. Get globe threshold from FP module 70
10.20. Set globe threshold from FP module 71
10.21. Enroll FP template into FP module 71
10.22. Get last time error condition 71
10.23. Verify FP template 72
10.24. Check if there is finger print above sensor 72
10.25. Get Max. template number 72
11. GET DEVICE ID 73
12. FLASH CONFIGURATION MANAGER - FLASHCONFIGMANAGER.DLL 74
12.1. Getting a Configuration Setting 74
12.2. Updating a Configuration Setting 74
12.3. Adding a Customer Configuration Setting 74
12.4. Deleting A Configuration Entry 75
12.5. Verifying an Entry's Value 75
12.6. Handling Errors 75
13. RS485 COMMUNICATION 76
13.1. Why RS485? 76
13.2. Windows APIs Used 76
13.3. Sample code 76
13.3.1. Opening the Com Port 76
13.3.2. Setting the Com Port Parameters 77
13.3.3. Writing Data To The Com Port 77
13.3.4. Reading Data From The Comm Port 78

13.3.5. Get error code 78

14. FUNCTION KEY SETTING ON REGISTRY 79

15. UPDATE NOTES 80

1. Introduction

1.1. Why SDK?

Microsoft had provided several standard SDKs for different WinCE platforms because there are different supporting modules, GUIs and components in each platform. There are un-predictable problem if user use improper SDK to compile your application. User should select proper SDK which match to target platform to develop application program. For Unitech terminal, we also provide SDK for each platform, you can get it from Technical Binder, or get the latest version from below URL.

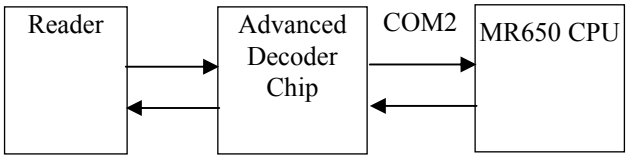
http://w3.tw.ute.com/pub/cs/SDK/MR650/MR650SDK.zip

You can also get sample source for Camera, Finger print and wave record/play from below URL

http://w3.tw.ute.com/pub/cs/software/sample_program/mr650/MR650_Sample_source.zip

1.2. How to download data from building RFID proximity reader

The major difference between the MR650 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 MR650. Inside the MR650 there is an advanced decoding chip to control SE900 laser engine and to handle barcode decoding. Below is system diagram for the MR650 barcode:



According to the above diagram, the MR650 communicates with Decoder Chip by mean of serial port COM2. Its communication parameter is fixed on 38400,N,8.1. Normally, the Decoder Chip is in sleep mode when COM2 is not activated. When COM2 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 MR650.

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 MR650'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, MR650 provide USI.DLL to let user control scanner input, symbologies setting and profile controlling. Please refer to for detail API lists.

1.3. COM definition for MR650

COM 1	Physical full RS232 port (ActiveSync)
COM 2	Scanner (Hamster)
COM 3	IrCom
COM 4	USB client
COM 5	RS485
COM 6	PCMCIA (if insert COM type PCMCIA card

1.4. Useful Sample program

You can get useful sample program for VC, C# and VB.NET from below URL
http://w3.tw.ute.com/pub/cs/software/sample_program/mr650/MR650_Sample_source.zip

It includes below functions' sample program

1. Finger print
2. Camera
3. RS485
4. Audio Wave record and play

2. USI.DLL - Unitech Scanner Interface DLL

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"
```

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 starts 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
msgID:	FALSE	: Disable Scanner

Return code:

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 code:

Always TRUE

2.5. Get error code

Function Description:

Returns the error code (SERR_***).

Function call:

```
DWORD USI_GetError();
```

Return code:

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

2.6. Return the system error code

Function Description:

Return 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);
```

Return code:

Return 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:

Retrieve the scan data into the buffer. Return 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
```

2.8. *Get length of scanned data*

Function Description:

Return 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 code:

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 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)

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

Return code:

BOOL: 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: (input)

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 code:

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 code:

TRUE if success, otherwise FALSE.

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 code:

TRUE = success
FALSE = error

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 code:

TRUE = success
FALSE = error

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

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 code:

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 code:

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 code:

TRUE = enabled
FALSE = disable

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 code:

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 code:

TRUE = success
FALSE = fail

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

```
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 code:

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 code:

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 code:

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

3. *Control command for decoder chip*

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	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...Si,BCC	Lh/Ll is command length. Si is in the range of 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. UnitechAPI.DLL

In MR650, Unitech create UnitechAPI.DLL to provide some special function calls which are different from standard Microsoft API. For example, RS232 is defined as host communication port with PC via ActiveSync, so it will automatically invoke ActiveSync program to do communication with PC when RS232 cable is plugged into MR650. However, it will make RS232 port useless if user want to connect MR650 with any device with RS232 interface. RS232Event.DLL provides function call for user to disable ActiveSync function over RS232 port to let user directly control RS232 port.

Unitech also provide several function to enable/disable several system icon and task bar. For WinCE system, it just like Windows OS platform, user can directly tap "Start" button from task bar to setup terminal or execute any application on WinCE terminal, so it mean that operator can change, modify or delete any setting. If system developer don't want operator to do any extra operation beside application, Unitech provide function call to provides ability to disable/enable task bar, keyboard and etc.

You can get demo program from MR650 technical binder zip files from \programming\UnitechAPI

4.1. Disable ActiveSync

Function Description:

After called this function, MR650 will not automatically execute ActiveSync program("repllog.exe") when user plug RS232 cable into MR650.

Function call:

BOOL RS232EventEnable (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.2. Enable ActiveSync

Function Description:

After called this function, MR650 will automatically execute ActiveSync program ("repllog.exe") again when user plug RS232 cable into MR650.

Function call:

BOOL RS232EventEnable (LPTSTR);

Parameter (Input):

String buffer and content should be "REPLLOG.EXE". If user assign other program, it will invoke user defined program rather than "REPLLOG.EXE"

Return code:

TRUE = OK
FALSE = File not found

4.3. Suspend

Function Description:

After called this function, MR650 will automatically suspend itself.

Function call:

void Suspend (void);

4.4. Disable TaskBar

Function Description:

This function will hide "TaskBar" and it doesn't like "Auto Hide" function which is set from Start Settings TaskBar. "TaskBar" can not be show again when tap button of LCD screen. It need to execute "Enable_TaskBar()" to enable it again.

Function call:

BOOL DisableTaskbar (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.5. Enable TaskBar

Function Description:

This function will show “TaskBar” again after “Disable_TaskBar()” was executed to hide taskbar.

Function call:

BOOL EnableTaskbar (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.6. Disable Desktop

Function Description:

This function will hide all icons on desktop, it mean that any short-cut or files cannot be accessed or executed.

Function call:

BOOL DisableDesktop (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.7. Enable Desktop

Function Description:

This function will show all icons which had already showed on desktop before executed DisableDesktop().

Function call:

BOOL EnsableDesktop (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.8. Disable toolbar on windows explorer

Function Description:

This function will hide windows explorer’s toolbar.

Function call:

BOOL DisableExploreToolbar (VOID);

Return code:

TRUE = OK
FALSE = Fail

4.9. Enable toolbar on windows explorer

Function Description:

This function will enable windows explorer’s toolbar again.

Function call:

BOOL EnsableExploreToolbar (VOID);

Return code:

TRUE = OK
FALSE = Fail

5. SysIOAPI.DLL (for Hardware related IO control API)

MR650 embeds several IO functions that may not be accessible through standard Windows API functions, such as following,

- Four Relay output controls
- Four optical isolated inputs
- Image
- Finger print module
- Proximity reader
- Other platform specific IO

Among these IO, image and finger print module will have individual section to explain detail function call. This section will show how to access relays, optical input, and other specific IO functions. Relay output and optical input can be accessed by two methods, through IO device control or DLL functions.

With IO control method, no external library necessary, applications use CreateFile() function to open IO control device and use DeviceIoControl() function to access IO.

With DLL method, a DLL named SysIOAPI.DLL, is embedded in system and corresponding library and include files are provided in SDK for linking with application.

Either method should work fine, but IO device method is more generic.

Those functions are,

IO device control method

BOOL IOC_GetRelayStatus(DWORD *)
BOOL IOC_SetRelay1(BOOL),
BOOL IOC_SetRelay2(BOO),
BOOL IOC_SetRelay3(BOOL),
BOOL IOC_SetRelay4(BOOL)

BOOL IOC_GetOpticalInput(DWORD *)

BOOL IOC_Module1PowerStatus(DWORD *)
BOOL IOC_PowerModule1(BOOL)

BOOL IOC_GetPCMCIA0Status(DWORD *)
BOOL IOC_EnablePCMCIA0(BOOL)

BOOL IOC_Camera_PowerOn()
BOOL IOC_Camera_PowerOff()
BOOL IOC_Camrea_Reset()

DLL method

DWORD GetRelayStatus(void)
BOOL SetRelay(DWORD, BOOL)
DWORD GetOpticalInputStatus(void)
BOOL SetModule1Power(BOOL);
UINT GetPCMCIASlotID(UINT)
void EnablePCMCIASlot1(UINT, BOOL);
BOOL CameraPowerOn(BOOL);
BOOL CameraReset(void);

5.1. IO Device Control Method

To use those function call, it is necessary to include “ioc_ioctl.h”. Please refer to below method and sample source.

5.1.1. Create IO Device

```
…
#include “ioc_ioctl.h”
…

HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L”IOC1:”, GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…

CloseHandle(gIOControlDriverHandle);
…
```

After IO device opened, IO functions can be accessed through DeviceIoControl(). If the device handle has the same name as gIOControlDriverHandle, then help macros defined in ioc_ioctl.h are ready to use.

In following examples, we assume IO device has been successfully opened already with handle name gIOControlDriverHandle as shown in above code excerpt. And we use help macros to access IO.

5.1.2. Close Device

Device has to be closed before application exit by calling, “CloseHandle(gIOControlDriverHandle)”.

5.2. Access Relays

5.2.1. Check Relay Status

Function Description:

To get current Relay status.

Function call:

BOOL IOC_GetRelayStatus(DWORD *pStatus);

Parameters(Output):

pStatus: DWORD * : Relay status

Return code:

TRUE = Success

FALSE = Unsupported

Note:

This macro returns relay status in pStatus, which is a point to a DWORD. Macro RELAY1_BITS… RELAY4_BITS define which bit in the *pStatus corresponds to which relay.

Example:

```
#include "ioc_ioctl.h"
…
DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…
gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE) {
// IO device not found, function not supported,
// error process
}
…

IOC_GetRelayStatus(&dwStatus);
if (dwStatus & RELAY1_BITS) {
 // relay 1 close
}
else {
      // relay 1 open
}

if (dwStatus & RELAY2_BITS) {
 // relay 2 close
}
else {
      // relay 2 open
}

if (dwStatus & RELAY3_BITS) {
 // relay 3 close
}
else {
      // relay 3 open
}

if (dwStatus & RELAY4_BITS) {
 // relay 4 close
}
else {
      // relay 4 open
}

CloseHandle(gIOControlDriverHandle);
```

5.2.2. Set Relay

Function Description:

There four macro defined for four relay output.

Function call:

BOOL IOC_SetRelay1(BOOL On);
BOOL IOC_SetRelay2(BOOL On);
BOOL IOC_SetRelay3(BOOL On);
BOOL IOC_SetRelay4(BOOL On);

Parameters(Input):

On: BOOL : TRUE for relay close
FALSE for relay open.

Return code:

TRUE = Success
FALSE = Unsupported

Example:

```
#include "ioc_ioctl.h"
…
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                     GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
IOC_SetRelay1(TRUE);     // close relay 1
IOC_SetRelay4(FALSE);    // open relay 4
…

CloseHandle(gIOControlDriverHandle);
```

5.3. Access Optical Isolated Input

5.3.1. Get Optical Isolated Input Status

Function Description:

Get optical isolated status.

Function call:

BOOL IOC_GetOpticalInput(DWORD *pStatus);

Parameters(Output):

pStatus: DWORD * : Optical status

Return code:

TRUE = Success

FALSE = Unsupported

Note:

This function is similar to IOC_GetRelayStatus(pStatus), but with optical input status returned. Macro OPTICAL1_BITS ⋯ OPTICAL4_BITS define which bit of *pStatus corresponds to which input.

Example:

```
#include "ioc_ioctl.h"
…
DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…
gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
     IOC_GetOpticalInput(&dwStatus);
if ( dwStatus & OPTICAL1_BITS) {
 // input 1 close
}
else {
       // input 1 open
}

if (dwStatus & OPTICAL 2_BITS) {
 // input 2 close
}
else {
       // input 2 open
}

if (dwStatus & OPTICAL 3_BITS) {
 // input 3 close
}
else {
       // input 3 open
}
```

5.3.2. Event for Optical Input Change

Function Description:

It is more effective if check input only when it changes status. When any input changes it status, a named event, IOCOpticalInputEvent, is generated. Applications can use a thread to catch this event for necessary tasks. An example thread as below,

```
…
void WINAPI IOInputthread()
{
  HANDLE gOpticalIOInputEvent = NULL;
  DWORD dwStatus;
  HANDLE gIOControlDriverHandle;
  …

  gIOControlDriverHandle = CreateFile(L”IOC1:”, GENERIC_READ |
                        GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
            NULL, OPEN_EXISTING, 0, 0);
  If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
  {
  // IO device not found, function not supported,
  // error process
  }
  …

  gOpticalIOInputEvent = CreateEvent(NULL, FALSE, FALSE,
  IOC_OPTICAL_INPUT_EVENT);
  if (gOpticalIOInputEvent == NULL)
  {
            // not supported
            // error process
  }

  while (1)
  {
            WaitForSingleObject(gOpticalIOInputEvent, INFINITE);

            IOC_GetOpticalInput(&dwStatus);
            // process input change tasks
  //…
  }
}
```

In ioc_ioctl.h, IOC_OPTICAL_INPUT_EVENT is defined as

#define IOC_OPTICAL_INPUT_EVENT _T("IOCOpticalInputEvent")

Be aware to release handles when program ends.

5.4. Get Back Cover Open/Close status

Function Description:

Back cover switch is beside UPS battery. This switch will be pressed (close) when back cover is installed into MR650 main body. And switch will be released when back cover is remove from MR650(Open). Application program can detect switch status to see if MR650' s back plate is remove or not for security purpose.

Function call:

BOOL IOC_GetCoverStatus(DWORD *pStatus);

Parameters(Output):

pStatus: DWORD * : Cover status = TRUE : Close
= FALSE : Open

Return code:

TRUE = Success
FALSE = Unsupported

Example:

Cover change event shares the same name as optical input, which is "IOCOpticalInputEvent". Applications can use a thread to catch this event for necessary tasks. An example thread as below,

```
void WINAPI IOCoverSwitchMonitor()
{
 HANDLE gCoverSwitchEvent = NULL;
 DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…

 gCoverSwitchEvent = CreateEvent(NULL, FALSE, FALSE,
IOC_OPTICAL_INPUT_EVENT);
 if (gCoverSwitchEvent == NULL)
 {
       // not supported
       // error process
 }

 while (1)
 {
       WaitForSingleObject(gCoverSwitchEvent, INFINITE);
       IOC_GetCoverStatus (&dwStatus);
       // process cover change tasks
//…
 }

 CloseHandle(gIOControlDriverHandle);
 CloseHandle(gCoverSwitchEvent);
…
```

In ioc_ioctl.h, IOC_OPTICAL_INPUT_EVENT is defined as

#define IOC_OPTICAL_INPUT_EVENT _T("IOCOpticalInputEvent")

Be aware to release handles when program ends.

5.5. Power Control

5.5.1. Get Module 1 Power Status

Function Description:

When system is operated on battery, power saving is necessary. Module 1 power control is assigned to finger printer module in MR650.

Function call:

BOOL IOC_Module1PowerStatus(DWORD *pStatus);

Parameters(Output):

pStatus: DWORD * : Optical status

Return code:

TRUE = Success

FALSE = Unsupported

Example:

```
DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L”IOC1:”, GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
…

IOC_Module1PowerStatus(&dwStatus);
if (dwStatus & MODULE1_POWER_BITS)
{
 // power is on
}
else {
 // power is off
}
…
CloseHandle(gIOControlDriverHandle);
…
```

5.5.2. Control Module 1 Power

Function Description:

When system is operated on battery, power saving is necessary. Module 1 power control is assigned to finger printer module in MR650.

Function call:

BOOL IOC_PowerModule1(BOOL On);

Parameters(Input):

pStatus: DWORD * : Optical status

Return code:

TRUE = Success

FALSE = Unsupported

Example:

```
#include  “ioc_ioctl.h”
…
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L”IOC1:”, GENERIC_READ |
                     GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
       NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…

IOC_PowerModule1(TRUE);        // power on module 1
IOC_PowerModule1(FALSE); // power off module 1
…

CloseHandle(gIOControlDriverHandle);
…
```

5.6. PCMCIA Control

MR650 provides a build-in CF slot. Using following macros to check/enable/disable this slot.

5.6.1. Check CF Slot

Function Description:

To check CF slot status.

Function call:

BOOL GetPCMCIA0Status(DWORD *pStatus);

Parameters(Output):

pStatus: DWORD * : CF slot enable/disable status

Return code:

TRUE = Success

FALSE = Unsupported

Example:

```
#include “ioc_ioctl.h”
…
DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L” IOC1:” , GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
…
 GetPCMCIA0Status(&dwStatus);
 if (dwStatus == 0)
 {
       // CF disabled
 }
 else
{
// CF enabled
}
CloseHandle(gIOControlDriverHandle);
…
```

5.6.2. Enable/Disable CF Slot

Function Description:

Enable/disable CF slot.

Function call:

BOOL IOC_EnablePCMCIA0(BOOL On);

Parameters(Input):

On: BOOL : CF slot status = TRUE : Enable
= FALSE : Disable

Return code:

TRUE = Success
FALSE = Unsupported

Example:

```
#include "ioc_ioctl.h"
…
DWORD dwStatus;
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                    GENERIC_WRITE,  FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
…
 GetPCMCIA0Status(&dwStatus);
 if (dwStatus == 0)
 {
       IOC_EnablePCMCIA0(TRUE);    //if  CF disabled, enable it
 }
 else
{
IOC_EnablePCMCIA0(FALSE);   //if CF enabled, disable it
}
CloseHandle(gIOControlDriverHandle);
…
```

5.7. Camera Control

5.7.1. Power on/off Camera

Function Description:

Power on/off Camera.

Function call:

BOOL IOC_Camera_PowerOn();
BOOL IOC_Camera_PowerOff();

Return code:

TRUE = Success
FALSE = Unsupported

5.7.2. Camera Reset

Function Description:

Reset Camera.

Function call:

BOOL IOC_Camera_Reset();

Return code:

TRUE = Success
FALSE = Unsupported

Example:

```
#include "ioc_ioctl.h"
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
                      GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
      NULL, OPEN_EXISTING, 0, 0);
If (gIOControlDriverHandle == INVALID_HANDLE_VALUE)
{
// IO device not found, function not supported,
// error process
}
…
…

IOC_Camera_PowerOn();
IOC_Camera_Reset();
…
IOC_Camera_PowerOff();
…

CloseHandle(gIOControlDriverHandle);
…
```

5.8. Watch Dog

5.8.1. Set time out value

Function Description:

Set a watchdog timeout value. After this function being called, it will not take effect until "StartWatchdog" is called.

Function call:

void SetWatchdogTimeout(DWORD mSecs);

Parameters(Input):

mSecs: DWORD : Time out value of milliseconds

5.8.2. Start the watchdog

Function Description:

Start the watchdog. System will reboot at the time defined by the last "SetWatchdogTimeout". Once watchdog is started, the hardware cannot stop it. So the only way to avoid system boot is to keep calling "SetWatchdogTimeout" to update the next watchdog timeout value.

Function call:

void StartWatchdog();

5.8.3. Pause the watchdog

Function Description:

Pause the watchdog. Actually hardware watchdog cannot be stopped. This function allows the user application to stop calling "SetWatchdogTimeout". Instead, a thread in the IOC driver will be calling "SetWatchdogTimeout" continuously, to avoid watchdog reboot. Once paused, the watchdog can be restarted by calling "StartWatchdog".

Function call:

void StartWatchdog();

Example:

```
#include "ioc_ioctl.h"
HANDLE gIOControlDriverHandle;
…

gIOControlDriverHandle = CreateFile(L"IOC1:", GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
       NULL, OPEN_EXISTING, 0, 0);

if(gIOControlDriverHandle != INVALID_HANDLE_VALUE)
{
  IOC_SetWatchdogTimeout(3000);
  …
…
IOC_StartWatchdog();
…
…
IOC_PauseWatchdog();
…
…
  CloseHandle(gIOControlDriverHandle);
}
else
{
  …
…
}
```

6. Camera Control - CameraDLL.DLL

This DLL is used to control MR650's internal CCD camera.

p.s. Camera' power on/off control functions are put on SysIOAPI.DLL because it is related to H/W function. Please refer to Section 5.6

6.1. Open Camera

Function Description:

To open Camera. This API should be called first before executed any other functions.

Function call:

BOOL OpenCamera();

Return code:

TRUE:	OK
FALSE:	Fail to open

6.2. Close Camera

Function Description:

To close Camera. This function must be called after using the camera.

Function call:

void ReleaseCamera();

6.3. Preview CCD video input

Function Description:

Start preview within a rectangle area as given in the parameters (using screen coordinates). Actual displaying area can be a little smaller than the rectangle for hardware reasons.

Function call:

void StartPreview(UINT left, UINT top, UINT width, UINT height);

Parameters(Input):

left:	UINT	: horizontal starting pixels point from left-top corner
top:	UINT	: vertical starting pixels point from left-top corner
width:	UINT	: horizontal pixel width
height:	UINT	: vertical pixel width

6.4. Stop Preview CCD video input

Function Description:

Stop preview.

Function call:

void StopPreview();

6.5. Capture image

Function Description:

Capture a photo with the specified resolution, must be called within StartPreview() and StopPreview().

Function call:

BOOL CaptureImage(LPCTSTR filename, UINT width, UINT height);

Parameters(Input):

filename:	LPCTSTR	: File name (with path) to store capture image
width:	UINT	: horizontal pixel width
height:	UINT	: vertical pixel width

Return code:

TRUE:	OK
FALSE:	Fail to capture

6.6. Get color key

Function Description:

Get drawing background color of the preview screen.

Function call:

DWORD GetColorKey();

Return code:

DWORD:	1st byte Red color value
	2nd byte Green color value
	3rd byte Blue color value

6.7. Set color key

Function Description:

Set drawing background color of the preview screen.

Function call:

void SetColorKey(UINT8 red, UINT8 green, UINT8 blue);

Parameters(Input):

red:	UINT8	: Red color value
green:	UINT8	: Green color value
blue:	UINT8	: Blue color value

6.8. Start video capture

Function Description:

Start video capture, must be called within StartPreview() and StopPreview().

Function call:

BOOL StartVideoCapture(LPCTSTR filename, UINT width, UINT height, UINT fps);

Parameters(Input):

filename:	LPCTSTR	: File name (with path) to store video(raw MPEG4)
width:	UINT	: horizontal pixel width
height:	UINT	: vertical pixel width
fps:	UINT	: frames per second

Return code:

TRUE:	OK
FALSE:	Fail to capture

6.9. End video capture

Function Description:

Stop video capture.

Function call:

BOOL EndVideoCapture();

Return code:

TRUE:	OK
FALSE:	Fail

6.10. Video playback function

The following 8 functions are for the video playback. After EndVideoCapture(), there will be a .m4v(raw mpeg4) video file created, it can be playbacked on the demo application using the following APIs, NOTICE THAT these function must be called after StopPreview().

6.10.1. Initiate playback

Function Description:

Setup initial condition for playback.

Function call:

BOOL InitializeDecoder(LPCTSTR filename, UINT left, UINT top, UINT width, UINT height);

Parameters(Input):

filename:	LPCTSTR	: File name (with path) to store video(raw MPEG4)
left:	UINT	: horizontal starting pixels point from left-top corner
top:	UINT	: vertical starting pixels point from left-top corner
width:	UINT	: horizontal pixel width
height:	UINT	: vertical pixel width

Return code:

TRUE = Success
FALSE = Fail

6.10.2. Release playback

Function Description:

Called after the playback to release the decoder.

Function call:

void UnInitializeDecoder();

6.10.3. Start playback

Function Description:

Start the playback. playback will be displayed in the same screen as the preview screen.

Function call:

BOOL StartPlayback();

Return code:

TRUE = Success
FALSE = Fail

6.10.4. Stop playback

Function Description:

Stop the playback.

Function call:

void StopPlayback();

6.10.5. Pause playback

Function Description:

Pause the playback.

Function call:

void PausePlayback();

6.10.6. Continue playback

Function Description:

Resume the playback from pause.

Function call:

void ContinuePlayback();

6.10.7. Check if playback

Function Description:

Check to see if play back is finished.

Function call:

BOOL IsPlaybackEnded();

Return code:

TRUE = Finished

6.11. Convert raw MPEG to standard MPEG

Function Description:

Convert .m4v(raw format) to .mp4(standard mpeg4 container), which is playable on quicktime, divx, xvid players.

Function call:

BOOL ConvertM4vToMP4(char* MP4File, LPCTSTR M4vFile);

Parameters(Input):

MP4File:	char *	: Standard MPEG 4
M4vFile:	LPCTSTR	: Raw MPEG 4 file to be play

Return code:

TRUE:	Success
FALSE:	Fail

7. Finger print control BIOIDDLL.DLL

Below API maybe useful for you to control MR650's UPEK finger print module.
Please download the sample program from below link.
http://w3.tw.ute.com/pub/cs/software/Sample_Program/MR650/UPEKDemo.zip

7.1. Power On/Off the Finger Print Module

Function Description:

To turn on/off the finger print module on MR650.

Function call:

BOOL PTPowerSwitch(BOOL bOn);

Parameters(Input):

bOn: BOOL : True for power on, false for power off

Return code:

BOOL: True for success, false for fail

7.2. Initialize Finger Print Library

Function Description:

Initialize the API library. Must be called before any other function.

Function call:

PT_STATUS PTInitialize(PT_MEMORY_FUNCS *pMemoryFuncs);

Parameters(Input):

pMemoryFuncs: PT_MEMORY_FUNCS : Structure pointer to the memory allocation and unallocation routines.

P.S. PT_MEMORY_FUNCS is a structure which is defined in BioIDTypes.h as below

```
typedef struct pt_memory_funcs {
        PT_MALLOC pfnMalloc;        /**< Memory allocating function */
        PT_FREE pfnFree;            /**< Memory freeing function */
} PT_MEMORY_FUNCS;
```

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.3. Uninitialize Finger Print Library

Function Description:

Uninitialize the API library. Must not be called while any connection is still open. There is usually no need to call this function.

Function call:

PT_STATUS PTTerminate(void);

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.4. Connect to Finger Print Module

Function Description:

Connect to the finger print module.

Function call:

PT_STATUS PTOpen(PT_CHAR *pszDsn, PT_CONNECTION *phConnection);

Parameters(Input):

pszDsn: PT_CHAR : Zero terminated ASCII string descriobing the TFM connection parameters, set to NULL.

Parameters(Output):

phConnection: PT_CONNECTION : Resulting connection handle.

P.S. PT_CONNECTION is the connection handle which is defined in BioIDTypes.h.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.5. Disconnect to Finger Print Module

Function Description:

Disconnect to the finger print module.

Function call:

PT_STATUS PTClose(PT_CONNECTION hConnection);

Parameters(Input):

phConnection: PT_CONNECTION : The connection handle.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.6. GUI Callback Function

Function Description:

Sets the address of the callback routine to be called if any called function involves displaying a biometric user interface.

Function call:

PT_STATUS PTSetGUICallbacks(PT_CONNECTION hConnection, PT_GUI_STREAMING_CALLBACK pfnGuiStreamingCallback, void *pGuiStreamingCallbackCtx, PT_GUI_STATE_CALLBACK pfnGuiStateCallback, void *pGuiStateCallbackCtx);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
pfnGuiStreamingCallback:	PT_GUI_STREAMING_CALLBACK	: A pointer to an application callback to deal with the presentation of biometric streaming data. Reserved for future used, currently not implemented. Use NULL for this parameter.
pGuiStreamingCallbackCtx:	void	:A generic pointer to context information provided by the application that will be presented on the callback. Reserved for future used, currently not implemented. Use NULL for this parameter.
pfnGuiStateCallback:	PT_GUI_STATE_CALLBACK	: A pointer to an application callback to deal with GUI state changes.
pGuiStateCallbackCtx:	void	: A generic pointer to context information provided by the application that will be presented on the callback

P.S. PT_GUI_STREAMING_CALLBACK and PT_GUI_STATE_CALLBACK are the point of callback function which is defined in BioIDTypes.h.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.7. Release Memory Block

Function Description:

Releases memory block using deallocation function passed to API by PTInitialize() call.

Function call:

void PTFree(void *Memblock);

Parameters(Input):

Memblock: void : The memory block point.

7.8. Get the Image of Living Finger

Function Description:

Scan the living finger and return the scanned image.

Function call:

PT_STATUS PTGrab(PT_CONNECTION hConnection, PT_BYTE byType, PT_LONG lTimeout,
PT_BOOL boWaitForAcceptableFinger,PT_DATA **ppGrabbedData,
PT_DATA *pSignData, PT_DATA **ppSignature
);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
byType:	PT_BYTE	: Requested type of the returned data. The data may be compressed, subsampled, etc. Use one of the PT_GRAB_TYPE_xxx values.
lTimeout:	PT_LONG	: Timeout in miliseconds. "-1" means default timeout.
boWaitForAcceptableFinger:	PT_BOOL	: If PT_TRUE, the function verifies the quality of the finger image and returns only when the finger quality would be acceptable for other biometric functions. If PT_FALSE, the function return immediately without verifying image quality.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppGrabbedData:	PT_DATA	: Address of the pointer, which will be set to point to the resulting grabbed data. The data has to be discarded by a call to PTFree().
ppSignature:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.9. Get the Image of Living Finger by Defined Window

Function Description:

Scan the live finger and return the scanned image. Only a defined window of the sensor will be returned.

Function call:

```
PT_STATUS PTGrabWindow(PT_CONNECTION hConnection, PT_BYTE byCompressionType,
                       PT_LONG lTimeout, PT_BOOL boWaitForAcceptableFinger,
                       PT_LONG lRows, PT_LONG lRowOffset, PT_LONG lRowDelta,
                       PT_LONG lCols, PT_LONG lColOffset, PT_LONG lColDelta,
                       PT_DATA **ppGrabbedData,  PT_DATA *pSignData, PT_DATA
                       **ppSignature);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
byCompressionType:	PT_BYTE	: Requested type of compression of the data. Use one of the PT_GRAB_COMPRESSION_TYPE_xxxx values.
lTimeout	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
boWaitForAcceptableFinger	PT_BOOL	: If PT_TRUE, the function verifies the quality of the finger image and returns only when the finger quality would be acceptable for other biometric functions. If PT_FALSE, the function returns immediately without verifying image quality.
lRows	PT_LONG	: Number of rows to grab. This is the number of actually grabbed rows, taking into account the subsampling factor. Example: If lRows=4, lRowOffset=10 and lRowDelta=2, the resulting image will contain rows 10, 12, 14 and 16. This is the number of actually grabbed rows, taking into account the subsampling factor.
lRowOffset	PT_LONG	: First row to grab. [0,0] = Upper left corner.
lRowDelta	PT_LONG	: Subsampling factor = number of rows to advance between the neighbour used rows. 1=Normal full image.
lCols	PT_LONG	: Number of columns to grab.
lColOffset	PT_LONG	: First column to grab. [0,0] = Upper left corner.
lColDelta	PT_LONG	: Subsampling factor = number of columns to advance between the neighbour used columns. 1=Normal full image.
pSignData	PT_DATA	:Reserved, use NULL.

Parameters(Output):

ppGrabbedData:	PT_DATA	: Address of the pointer, which will be set to point to the resulting grabbed data. The data has to be discarded by a call to PTFree().
ppSignature:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.10. Save Living Finger Print Template to Module

Function Description:

Scan the live finger once or several times, depending on the session settings and combine the images into one enrollment template. The last template obtained through PTEnroll will be remembered throughout the session and can be used by biometric matching functions. This function can call GUI callbacks.

Function call:

PT_STATUS PTEnroll(PT_CONNECTION hConnection, PT_BYTE byPurpose, PT_INPUT_BIR *pStoredTemplate, PT_BIR **ppNewTemplate, PT_LONG *plSlotNr, PT_DATA *pPayload, PT_LONG lTimeout, PT_BIR **ppAuditData, PT_DATA *pSignData, PT_DATA **ppSignature);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
byPurpose:	PT_BYTE	: Purpose of the enrollment. Use one of the PT_PURPOSE_xxxx values.
pStoredTemplate:	PT_INPUT_BIR	: Reserved, use NULL.
pPayload:	PT_DATA	: Data to be embedded into the resulting template. Payload data is an output parameter from PTVerify and PTVerifyEx when successful match is achieved.
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppNewTemplate:	PT_ PT_BIR	: Address of the pointer, which will be set to point to the resulting template (BIR). The template has to be discarded by a call to PTFree(). If the template should be stored only in TFM's non-volatile memory, leave this parameter NULL.
plSlotNr:	PT_LONG	: Pointer to a variable which receives slot number (0..N-1) where the template was stored. If the value is NULL, template is not stored on TFM.
ppAuditData:	PT_BIR	: Reserved, use NULL.
ppSignature	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.11. Finger Print Verify

Function Description:

Scan the live finger or use the last captured finger data and try to match it against the supplied enrollment template. If the functions scans live finger, the template obtained will be remembered througout the session and can be used by other biometric matching functions. This function can call GUI callbacks (unless boCaptured is FALSE);

Function call:

```
PT_STATUS PTVerify(PT_CONNECTION hConnection, PT_LONG *plMaxFARRequested,
                   PT_LONG *plMaxFRRRequested, PT_BOOL *pboFARPrecedence,
                   PT_INPUT_BIR *pStoredTemplate, PT_BIR **ppAdaptedTemplate,
                   PT_BOOL *pboResult, PT_LONG *plFARAchieved, PT_LONG
                   *plFRRAchieved, PT_DATA **ppPayload, PT_LONG lTimeout,
                   PT_BOOL boCapture, PT_BIR **ppAuditData, PT_DATA
                   *pSignData,PT_DATA **ppSignature);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
plMaxFARRequested:	PT_LONG	: Max. FAR requested by the caller. Value of this parameter is currently ignored.
plMaxFRRRequested:	PT_LONG	: Max. FRR requested by the caller. Optional, can be NULL. Value of this parameter is currently ignored.
pboFARPrecedence:	PT_BOOL	: If both FAR and FRR are provided, this parameter decides which of them takes precedence: PT_TRUE -> FAR, PT_FALSE -> FRR. Value of this parameter is currently ignored.
pStoredTemplate:	PT_INPUT_BIR	: The template to be verified against - BIR data or one of the predefined handles.
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
boCapture:	PT_BOOL	: If PT_TRUE, PTVerify at first captures live fingerprint. If PT_FALSE, result of the last finger capturing function (e.g. PTCapture or PTEnroll) will be used.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppAdaptedTemplate:	PT_BIR	: Reserved, use NULL.
pboResult:	PT_BOOL	: The result: Match/no match
plFARAchieved:	PT_LONG	: Reserved, use NULL.
plFRRAchieved:	PT_LONG	: Reserved, use NULL.
ppPayload:	PT_DATA	: Address of the pointer, which will be set to point to the payload data, originally embedded in the @c pStoredTemplate. Payload data is available only when successful match is achieved.
ppAuditData:	PT_BIR	: Reserved, use NULL.
ppSignature:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.12. Finger Print Verify Ex

Function Description:

Scan the live finger or use the last captured finger data and try to match it against the set of supplied enrollment templates. If the function scans live finger, the template obtained will be remembered throughout the session and can be used by other biometric matching functions. Return the index of the best matching template or -1 if no match.

Function call:

```
PT_STATUS PTVerifyEx(PT_CONNECTION hConnection, PT_LONG
                     *plMaxFARRequested,PT_LONG *plMaxFRRRequested, PT_BOOL
                     *pboFARPrecedence, PT_INPUT_BIR *pStoredTemplates, PT_BYTE
                     byNrTemplates, PT_BIR **ppAdaptedTemplate, PT_SHORT *pshResult,
                     PT_LONG *plFARAchieved, PT_LONG *plFRRAchieved, PT_DATA
                     **ppPayload, PT_LONG lTimeout, PT_BOOL boCapture, PT_BIR
                     **ppAuditData, PT_DATA *pSignData, PT_DATA **ppSignature);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
plMaxFARRequested:	PT_LONG	: Max. FAR requested by the caller. Value of this parameter is currently ignored.
plMaxFRRRequested:	PT_LONG	: Max. FRR requested by the caller. Optional, can be NULL. Value of this parameter is currently ignored.
pboFARPrecedence:	PT_BOOL	: If both FAR and FRR are provided, this parameter decides which of them takes precedence: PT_TRUE -> FAR, PT_FALSE -> FRR. Value of this parameter is currently ignored.
pStoredTemplates:	PT_INPUT_BIR	: An array of templates to be verified against - BIR data or one of the predefined handles.
byNrTemplates:	PT_BYTE	: Number of templates in pStoredTemplates
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
boCapture:	PT_BOOL	: If PT_TRUE, PTVerifyEx at first captures live fingerprint. If PT_FALSE, result of the last finger capturing function (e.g. PTCapture or PTEnroll) will be used.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppAdaptedTemplate:	PT_BIR	: Reserved, use NULL.
pshResult:	PT_SHORT	: The result: The zero-based index of the best matching template or -1 if no match.
plFARAchieved:	PT_LONG	: Reserved, use NULL.
plFRRAchieved:	PT_LONG	: Reserved, use NULL.
ppPayload:	PT_DATA	: Address of the pointer, which will be set to point to the payload data, originally embedded in the pStoredTemplate. Payload data is available only when successful match is achieved.
ppAuditData:	PT_BIR	: Reserved, use NULL.
ppSignatureIpConfig:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.13. Verify Finger Print with All Templates in the Module

Function Description:

Scan the live finger or use the last captured finger data and try to match it against the templates stored in TFM's non-volatile memory. If the function scans live finger, the template obtained will be remembered throughout the session and can be used by other biometric matching functions. Return the slot of the best matching template or -1 if no match.

This is an extension to BioAPI. Its main purpose is to be able to match user's finger against all the enrolled templates in the TFM's database, but without the complexity of the heavy-weight BioAPI database mechanism. This is not a real one-to-many matching, but one-to-few matching.

Function call:

```
PT_STATUS PTVerifyAll(PT_CONNECTION hConnection, PT_LONG *plMaxFARRequested,
                      PT_LONG *plMaxFRRRequested, PT_BOOL *pboFARPrecedence,
                      PT_BIR **ppAdaptedTemplate, PT_LONG *plResult, PT_LONG
                      *plFARAchieved, PT_LONG *plFRRAchieved, PT_DATA **ppPayload,
                      PT_LONG lTimeout, PT_BOOL boCapture, PT_BIR **ppAuditData,
                      PT_DATA *pSignData, PT_DATA **ppSignature);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
plMaxFARRequested:	PT_LONG	: Max. FAR requested by the caller. Value of this parameter is currently ignored.
plMaxFRRRequested:	PT_LONG	: Max. FRR requested by the caller. Optional, can be NULL. Value of this parameter is currently ignored.
pboFARPrecedence:	PT_BOOL	: If both FAR and FRR are provided, this parameter decides which of them takes precedence: PT_TRUE -> FAR, PT_FALSE -> FRR. Value of this parameter is currently ignored.
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
boCapture:	PT_BOOL	: If PT_TRUE, PTVerifyEx at first captures live fingerprint. If PT_FALSE, result of the last finger capturing function (e.g. PTCapture or PTEnroll) will be used.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppAdaptedTemplate:	PT_BIR	: Reserved, use NULL.
pshResult:	PT_ LONG	: The result: The zero-based index of the best matching template or -1 if no match.
plFARAchieved:	PT_LONG	: Reserved, use NULL.
plFRRAchieved:	PT_LONG	: Reserved, use NULL.
ppPayload:	PT_DATA	: Address of the pointer, which will be set to point to the payload data, originally embedded in the pStoredTemplate. Payload data is available only when successful match is achieved.
ppAuditData:	PT_BIR	: Reserved, use NULL.
ppSignatureIpConfig:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.14. Detect Finger Print

Function Description:

Verifies, if there is a finger on the sensor. If Timeout is nonzero, the function waits for the specified time interval until the required conditions are met.

Function call:

PT_STATUS PTDetectFingerEx(PT_CONNECTION hConnection, PT_LONG lTimeout,
PT_DWORD dwFlags);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout. "0" is an acceptable value for this function, it means "test if the current state meats the required conditions".
dwFlags:	PT_DWORD	: Bit mask, determining the behavior of the function and the conditions for which the function waits.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.15. Store Finger Print Tamelate to Module

Function Description:

Stores given fingerprint template in the selected slot in the non-volatile memory of the TFM. If pTemplate is NULL, it only clears the slot.

Function call:

PT_STATUS PTStoreFinger(PT_CONNECTION hConnection,PT_INPUT_BIR *pTemplate,
PT_LONG *plSlotNr);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
pTemplate	PT_INPUT_BIR	: Template (BIR) to be stored in the template cache.

Parameters(Output):

plSlotNr:	PT_LONG	: Pointer to a variable which receives slot number (0..N-1) where the template was stored. If the value is NULL, the template is not stored.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.16. Delete the Finger Print in the Module

Function Description:

Deletes fingerprint template stored in the selected slot in the non-volatile memory of the FM.

Function call:

PT_STATUS PTDeleteFinger(PT_CONNECTION hConnection, PT_LONG lSlotNr);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
lSlotNr:	PT_LONG	: Number of slot to delete (0..N-1)

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.17. Remove All Finger Print in Module

Function Description:

Deletes all fingerprint templates stored in slots in the non-volatile memory of the FM.

Function call:

PT_STATUS PTDeleteAllFingers(PT_CONNECTION hConnection);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.18. Store Finger Print to Module

Function Description:

Assign an additional application data to a finger template stored in FM's non-volatile memory.

Function call:

PT_STATUS PTSetFingerData(PT_CONNECTION hConnection,PT_LONG lSlotNr, PT_DATA *pFingerData);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
lSlotNr:	PT_LONG	: The slot number of the template to be associated with data.
pFingerData:	PT_DATA	: The data to be stored together with the template. If the data length is zero, the application data associated with given fingerprint will be deleted

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.19. Read Finger Print From Module

Function Description:

Read the additional appliciation data associated with a finger template stored in FM's non-volatile memory.

Function call:

PT_STATUS PTGetFingerData(PT_CONNECTION hConnection,PT_LONG lSlotNr, PT_DATA **ppFingerData);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
lSlotNr:	PT_LONG	: The slot number of the template whose application data should be read.

Parameters(Output):

ppFingerData:	PT_DATA	: Address of the pointer, which will be set to point to the application data associated with given fingerprint. If no data are associated with the fingerprint, the result will be a data block with zero length. The data has to be freed by a call to PTFree.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.20. List All Finger Print in the Module

Function Description:

The function set the globe verification value to FP module.

Function call:

PT_STATUS PTListAllFingers(PT_CONNECTION hConnection, PT_FINGER_LIST **ppFingerList);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.

Parameters(Output):

ppFingerList:	PT_FINGER_LIST	: Address of the pointer, which will be set to point to the list of stored fingerprints and their associated data. The data has to be freed by a call to PTFree.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.21. Get the Finger Print Template from Specific Slot

Function Description:

Read template data stored in given slot.

Function call:

PT_STATUS PTLoadFinger(PT_CONNECTION hConnection,PT_LONG lSlotNr,PT_BOOL boReturnPayload, PT_BIR **ppStoredTemplate);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
lSlotNr:	PT_LONG	: Number of slot from which the template should be read
boReturnPayload:	PT_BOOL	: If TRUE then the template is returned with its payload.

Parameters(Output):

ppStoredTemplate:	PT_BIR	: Address of the pointer, which will be set to point to the loaded template (BIR). The template has to be discarded by a call to PTFree().

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.22. Get Module Information

Function Description:

Return a set of information about the connection and the TFM, including the version of TFM.

Function call:

PT_STATUS PTInfo(PT_CONNECTION hConnection, PT_INFO **ppInfo);

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.

Parameters(Output):

ppInfo:	PT_INFO	: Address of the pointer, which will be set to point to the structure with TFM/ESS information. The structure has to be freed by a call to PTFree.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.23. Set the Session Parameter of Module

Function Description:

Set the session parameters of the TFM. The parameters influence especially the behavior of the biometric functions - e.g. should we use the advanced or the standard templates etc.

Function call:

```
PT_STATUS PTSetSessionCfgEx(PT_CONNECTION hConnection,PT_WORD wCfgVersion,
                            PT_WORD wCfgLength,PT_SESSION_CFG *pSessionCfg);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
wCfgVersion:	PT_WORD	: Version of the configuration data. Use the constant PT_CURRENT_SESSION_CFG
wCfgLength:	PT_WORD	: Length of the configuration data
pSessionCfg:	PT_SESSION_CFG	: Session configuration to be set

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.24. Get the Session Parameter of Module

Function Description:

Get the current session parameters of the TFM. The parameters influence especially the behavior of the biometric functions - e.g. should we use the advanced or the standard templates etc.

Function call:

```
PT_STATUS PTGetSessionCfgEx(PT_CONNECTION hConnection,PT_WORD wCfgVersion,
                            PT_WORD *pwCfgLength, PT_SESSION_CFG **ppSessionCfg);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
wCfgVersion:	PT_WORD	: Requested version of the configuration data

Parameters(Output):

pwCfgLength:	PT_WORD	: Pointer to the length of the received configuration data
ppSessionCfg:	PT_SESSION_CFG	: Returned session configuration

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.25. Get Remaining EEPROM Memory in the Module

Function Description:

Returns the size in bytes of the remaining EEPROM memory on the TFM available for application's usage.

Function call:

```
PT_STATUS PTGetAvailableMemory(PT_CONNECTION hConnection,PT_DWORD dwType,
                               PT_DWORD *pdwAvailableMemory);
```

Parameters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
dwType:	PT_DWORD	: Requested type of memory (see values PT_MEMTYPE_xxxx)

Parameters(Output):

pdwAvailableMemory:	PT_DWORD	: Returned size of remaining EEPROM memory

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

7.26. Get Living Finger Print Template

Function Description:

Scan the live finger and process it into a template. The last template obtained through PTCapture will be remembered throughout the session and can be used by biometric matching functions. In addition it can be optionally returned to the caller. This function can call GUI callbacks.

Function call:

```
PT_STATUS PTCapture(PT_CONNECTION hConnection, PT_BYTE byPurpose, PT_BIR
                    **ppCapturedTemplate, PT_LONG lTimeout, PT_BIR **ppAuditData,
                    PT_DATA *pSignData, PT_DATA **ppSignature);
```

meters(Input):

hConnection:	PT_CONNECTION	: The connection handle.
byPurpose:	PT_BYTE	: Requested type of memory (see values PT_MEMTYPE_xxxx)
lTimeout:	PT_LONG	: Timeout in milliseconds. "-1" means default timeout.
pSignData:	PT_DATA	: Reserved, use NULL.

Parameters(Output):

ppCapturedTemplate:	PT_DWORD	: Returned size of remaining EEPROM memory
ppAuditData:	PT_BIR	: Reserved, use NULL.
ppSignature:	PT_DATA	: Reserved, use NULL.

Return code:

PT_STATUS: The status code which is defined in BioIDError.h.

8. Useful function call - without include SysIOAPI.DLL

Below API maybe useful for you to control MR650.

8.1. 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 );
}
```

9. Mifare Reader Library

This library “MifareDll.dll” is used to control Mifare reader on MR650.
Please download the sample program from below link.
http://w3.tw.ute.com/pub/cs/software/Sample_Program/MR650/MifareDemo.zip

9.1. Connect to Mifare Reader

Function Description:

To create a connection with the reader before control it.

Function call:

int Mifare_Connect ();

Parameters:

None.

Return code:

Please refer to chapter 9.23.

9.2. Disconnect with Mifare Reader

Function Description:

Close the connection with reader.

Function call:

int Mifare_Disconnect ();

Parameters:

None.

Return code:

Please refer to chapter 9.23.

9.3. Check the connection with Mifare Reader

Function Description:

To check the connection with Mifare reader.

Function call:

BOOL Mifare_IsConnect ();

Parameters:

None.

Return code:

TRUE:	Connected.
FALSE:	Disconnected

9.4. Firmware Version

Function Description:

To get the firmware version of the reader.

Function call:

int Mifare_FirmwareVersion (char *szVersion, int *nLen);

Parameters:

szVersion:	[Out]	Point to the buffer that receive the version number.
nLen:	[In]	The size of the parameter szVersion.
	[Out]	Return the data length of szVersion.

Return code:

Please refer to chapter 9.23.

9.5. Write Key to EEPROM

Function Description:

Save the key to EEPROM.

Function call:

int Mifare_WriteKeyToEEPROM (int nKeyMode, int nSector, char *szKey);

Parameters:

nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B
nSector:	[In]	The sector number to save key.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.6. Load Key from EEPROM

Function Description:

Load the key from EEPROM.

Function call:

int Mifare_LoadKeyFromInternal (int nKeyMode, int nSector);

Parameters:

nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to load key.

Return code:

Please refer to chapter 9.23.

9.7. Load Key with User Defined

Function Description:

To load user defined key.

Function call:

int Mifare_LoadKeyWithUserDefine(char *szKey);

Parameters:

szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.8. Read Card's Block Data

Function Description:

Read the data form card's block.

Function call:

int Mifare_ReadCardData(char *szData, int *nLen, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

szData:	[Out]	Point to the buffer that receive the data
nLen:	[In]	The size of the parameter szData.
	[Out]	Return the data length of szData.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to read data.
nBlock:	[In]	The block number to read data.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.9. Write Card's Block Data

Function Description:

Write data to card's block.

Function call:

int Mifare_WriteCardData(char *szData, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

szData:	[In]	Point to the buffer containing the data to write.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to read data.
nBlock:	[In]	The block number to read data.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.10. Read Card's Value

Function Description:

Read the data form card's block with value type.

Function call:

int Mifare_ReadValue(UINT *nValue, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

nValue:	[Out]	Point to the buffer that receive the data.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to read value.
nBlock:	[In]	The block number to read value.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.11. Write Card's Value

Function Description:

Write data to card's block with value type.

Function call:

int Mifare_WriteValue(UINT nValue, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

nValue:	[In]	The value which be written to the card.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to write value.
nBlock:	[In]	The block number to write value.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.12. Increase Value

Function Description:

Increase the value in the card.

Function call:

int Mifare_IncrementValue(UINT nValue, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

nValue:		The value which be increased to the card.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to increased value.
nBlock:	[In]	The block number to increased value.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.13. Decrease Value

Function Description:

Decrease the value in the card.

Function call:

int Mifare_DecrementValue(UINT nValue, int nKeyMode, int nSector, int nBlock, char *szKey);

Parameters:

nValue:		The value which be decreased to the card.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to decreased value.
nBlock:	[In]	The block number to decreased value.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.14. Read Sector Data

Function Description:

Read a sector data in the card and it will return 3 blocks(48 bytes) data.

Function call:

int Mifare_ReadSectorData(char *szData, int *nLen, int nKeyMode, int nSector, char *szKey);

Parameters:

szData:	[Out]	Point to the buffer that receive the data
nLen:	[In]	The size of the parameter szData.
	[Out]	Return the data length of szData.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to read data.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.15. Read Multi Sectors Data

Function Description:

Read multi sectors data in the card and it will return 3 blocks(48 bytes) data with each sector.

Function call:

int Mifare_ReadMultiSectorData(char *szData, int *nLen, int nKeyMode, int nSector, int nSectorNum, char *szKey);

Parameters:

szData:	[Out]	Point to the buffer that receive the data
nLen:	[In]	The size of the parameter szData.
	[Out]	Return the data length of szData.
nKeyMode:	[In]	The key mode, 0 for Key A, 1 for Key B.
nSector:	[In]	The sector number to read data.
nSectorNum:	[In]	The number of sectors to be read.
szKey:	[In]	The key value for 12 characters.

Return code:

Please refer to chapter 9.23.

9.16. Reader Serial Number

Function Description:

Read reader's serial number.

Function call:

int Mifare_ReaderSN(char *szSN, int *nLen);

Parameters:

szSN:	[Out]	Point to the buffer that receive the serial number.
nLen:	[In]	The size of the parameter szSN.
	[Out]	Return the data length of szSN.

Return code:

Please refer to chapter 9.23.

9.17. Start Read Card Serial Number - I

Function Description:

Start read card serial number and if reader gets a data then the library will create an event "EVENT_MIFARE_GET_DATA".

Function call:

int Mifare_StartRead();

Parameters:

None.

Return code:

Please refer to chapter 9.23.

9.18. Start Read Card Serial Number - II

Function Description:

Start read card serial number and if reader gets a data then the library will create an event "EVENT_MIFARE_GET_DATA".

Function call:

int Mifare_StartReadEx(int nReadCycle);

Parameters:

nReadCycle : [In]The sleep milliseconds between two inventory..

Return code:

Please refer to chapter 9.23.

9.19. Stop Read Card Serial Number

Function Description:

Stop read card serial number.

Function call:

int Mifare_StopRead();

Parameters:

None.

Return code:

Please refer to chapter 9.23.

9.20. Get Card Serial Number

Function Description:

To get card serial number after get the event "EVENT_MIFARE_GET_DATA".

Function call:

int Mifare_GetDataEx(char *szData, int *nLen, BOOL bReversed);

Parameters:

szData:	[Out]	Point to the buffer that receive the data.
nLen:	[In]	The size of the parameter szData.
	[Out]	Return the data length of szData.
bReversed:	[In]	Reverse the card ID.

Return code:

Please refer to chapter 9.23.

9.21. Check Reading

Function Description:

To check the API "Mifare_StartRead()" is running or not.

Function call:

BOOL Mifare_IsReading();

Parameters:

None.

Return code:

TRUE:	Reading.
FALSE:	Not reading.

9.22. Get Library Version

Function Description:

To get the library's version.

Function call:

int Mifare_LibraryVersion(char *szVersion, int *nLen) ;

Parameters:

szVersion:	[Out]	Point to the buffer that receive the data
nLen:	[In]	The size of szVersion.
	[Out]	Point to the number of receive data length.

Return code:

Please refer to chapter 9.23.

9.23. Error Code

Name	Value	Description
ERR_SUCCESS	0X0000	The API works Successfully
ERR_TIMEOUT	0X0001	No response from reader.
ERR_FAIL	0X0002	Un-know fail.
ERR_PARAMETER_ERROR	0X0003	Request Parameter error
ERR_NO_DATA	0X0004	Does not get data
ERR_BUFFER_TOO_SMALL	0X0005	The buffer size of parameter is too small to get data.
ERR_WRITE_KEY_TO_EEPROM_FAIL	0X0006	Write key to EEPROM fail
ERR_LOAD_KEY_FAIL	0X0007	Load key fail
ERR_WRITE_CARD_ERROR	0X0008	Card write data error
ERR_WRITE_VALUE_ERROR	0X0009	Card write value error
ERR_INCREMENT_ERROR	0X000A	Increase value error
ERR_DECREMENT_ERROR	0X000B	Decrease value error
ERR_READING	0X000C	The API Mifare_StartRead is working, please stop it then execute this API again.
ERR_WRITE	0X0EEE	Send command fail
ERR_READ	0X0FFF	Get response fail.

10. Finger print control BIOIDDLL.DLL(Bioscrypt)

Below API maybe useful for you to control MR650's Bioscrypt finger print module.

10.1. Start Finger print function

Function Description:

Start MR650 finger print function, include allocate and initialize COM port data structure.

Function call:

LPCOMMDATA BioID_Create(void);

Return code:

LPCOMMDATA structure

P.S. LPCOMMDATA is a structure which is defined on BIOIDDLL.h as below

```
typedef struct _tagCOMMDATA {
   BYTE        IsConnected;
   HANDLE      hCom;                  // COM port handle
   BYTE        Port;                  // always COM 6 on MR650
} COMMDATA, FAR* LPCOMMDATA;
```

10.2. Stop Finger print function

Function Description:

Stop MR650 finger print function, include deallocate COM port data.

Function call:

void BioID_Destroy (LPCOMMDATA CommData);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

10.3. Connect to Finger print module

Function Description:

Connect to FP's COM port using the parameters given by lpCommData.

Function call:

WORD BioID_Connect(LPCOMMDATA CommData);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

Return code:

WORD

10.4. Disconnect Finger print module

Function Description:

Disconnect FP from COM port.

Function call:

void BioID_Disconnect(LPCOMMDATA CommData);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

10.5. Setup Finger print module communication type

Function Description:

Setup FP module communication port parameter.

Function call:

WORD BioID_SetupEx(LPCOMMDATA CommData, WORD Port, DWORD BaudRate, WORD StopBits, WORD Parity);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
Port:	WORD	: Module's port (always 1 for MR650)
BaudRate:	DWORD	: Module's baud rate
StopBits:	WORD	: Stop bit
Parity:	WORD	: Parity

Return code:

WORD: Always TRUE

10.6. Get module's communication type

Function Description:

Get FP module's communication port type(1 for RS232, 2 for RS485) and save in lpConfig->Type.

Function call:

WORD BioID_GetType(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Parameters(Output):

lpConfig:	LPCONFIG	: Below is definition for LPCONFIG

```
typedef struct _tagCONFIG {
   WORD      Port,     // 1 => Com1, 2 => Com2
   Type,               // 1 => RS232, 2 => RS485
   Baud;               // 1 => CBR_9600
                       // 2 => CBR_19200
                       // 3 => CBR_38400
                       // 4 => CBR_57600
                       // 5 => CBR_115200
}
```

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.7. Set module's communication type

Function Description:

Set FP module's communication port type(1 for RS232, 2 for RS485).

Function call:

WORD BioID_SetType(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

lpConfig: LPCONFIG : Below is definition for LPCONFIG

```
typedef struct _tagCONFIG {
   WORD     Port,    // 1 => Com1, 2 => Com2
   Type,             // 1 => RS232, 2 => RS485
   Baud;             // 1 => CBR_9600
                     // 2 => CBR_19200
                     // 3 => CBR_38400
                     // 4 => CBR_57600
                     // 5 => CBR_115200
}
```

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.8. Get FP module's baudrate

Function Description:

Get FP module's baudrate and save into lpConfig->Baud.

Function call:

WORD BioID_GetBaud(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

Parameters(Output):

lpConfig: LPCONFIG : refer to BioID_GetType()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.9. Set FP module's baudrate

Function Description:

Set FP module's baudrate.

Function call:

WORD BioID_SetBaud(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData: LPCOMMDATA : refer to BioID_Create()

lpConfig: LPCONFIG : refer to BioID_GetType()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.10. Get FP module's Aux Port baudrate

Function Description:

Get FP module's Aux port baudrate save into lpConfig->Baud.

Function call:

WORD BioID_GetBaudAux(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Parameters(Output):

lpConfig:	LPCONFIG	: refer to BioID_GetType()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.11. Set FP module's Aux Port baudrate

Function Description:

Set FP module's Aux port baudrate.

Function call:

WORD BioID_SetBaud(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
lpConfig:	LPCONFIG	: refer to BioID_GetType()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.12. Set FP module's Aux Port baudrate

Function Description:

Set FP module's Aux port baudrate.

Function call:

WORD BioID_SetBaud(LPCOMMDATA CommData, LPCONFIG lpConfig);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
lpConfig:	LPCONFIG	: refer to BioID_GetType()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.13. Get FP version

Function Description:

Get firmware version and save in lpMsg as Unicode string.

Function call:

WORD BioID_Version (LPCOMMDATA CommData, LPWSTR lpMsg);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Parameters(Output):

lpMsg:	LPWSTR	: version number

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.14. Initialize FP directory

Function Description:

Initialize directory cursor to the first record.

Function call:

WORD BioID_Dir_Init(LPCOMMDATA CommData);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.15. Read FP directory

Function Description:

Read the next record and save it in *Id and *Index.

Function call:

WORD BioID_Dir_Read (LPCOMMDATA CommData, DWORD* Id, WORD* Index);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Parameters(Output):

Id:	DWORD *	: ID number (-1 if invalid)
Index:	WORD *	: index number (-1 if invalid)

Return code:

WORD:	130	(=CmdErrTime)	Timeout
	7	(=CmdNO)	Error occurred during receiving data
	6	(=CmdYES)	Success.

10.16. Get template from FP module

Function Description:

Get finger print template from module's memory.

Function call:

WORD BioID_UpLoad(LPCOMMDATA CommData, DWORD Id, WORD Index, LPDWORD lpdwSize, LPBYTE lpTemplate);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
Id:	DWORD	: ID number
Index:	WORD	: index number

Parameters(Output):

lpdwSize:	LPDWORD	: Template buffer size
lpTemplate:	LPBYTE	: Template buffer

Return code:

WORD:	129 (=CmdErrLen)	if record length is greater than *lpdwSize.
	137 (=CmdMemoSysErr)	if memory allocation failed.
	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.17. Send template to FP module

Function Description:

Send one finger print template to module's memory.

Function call:

WORD BioID_DnLoad (LPCOMMDATA CommData, DWORD dwSize, LPBYTE lpTemplate);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
dwSize:	DWORD	: Template buffer size
lpTemplate:	LPBYTE	: Template buffer

Return code:

WORD:	137 (=CmdMemoSysErr)	if memory allocation failed.
	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.18. Remove template from FP module

Function Description:

Remove one finger print template from module's memory.

Function call:

WORD BioID_Remove (LPCOMMDATA CommData, DWORD Id, WORD Index);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
Id:	DWORD	: ID number
Index:	WORD	: index number

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.19. Get globe threshold from FP module

Function Description:

The function gets the globe verification value from FP module.

Function call:

WORD BioID_GetThresh(LPCOMMDATA CommData, LPBYTE lpThresh);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Parameters(Output):

lpThresh:	LPBYTE	: 1	very high security
		2	high security
		3	medium security
		4	low security
		5	very low security

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.20. Set globe threshold from FP module

Function Description:

The function set the globe verification value to FP module.

Function call:

WORD BioID_SetThresh(LPCOMMDATA CommData, BYTE nThresh);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
nThresh:	BYTE	: 1 very high security
		2 high security
		3 medium security
		4 low security
		5 very low security

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.21. Enroll FP template into FP module

Function Description:

Enroll finger print template into module.

Function call:

WORD BioID_Enroll(LPCOMMDATA CommData, DWORD Id, LPDWORD lpdwQuality, LPDWORD lpdwContent);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
Id:	DWORD	: ID number

Parameters(Output):

lpdwQuality:	LPDWORD	: 0 ~ 100
lpdwContent:	LPDWORD	: 0 ~ 100

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.22. Get last time error condition

Function Description:

Get the latest error condition.

Function call:

int BioID_GetLastError(LPCOMMDATA lpCommData);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()

Return code:

WORD:	129 (=CmdErrLen)	if record length is greater than *lpdwSize.
	137 (=CmdMemoSysErr)	if memory allocation failed.
	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.23. Verify FP template

Function Description:

The function requests to verify the fingerprint against the template(s) with the specified ID.

Function call:

WORD BioID_Verify(LPCOMMDATA CommData, DWORD Id, LPDWORD lpdwVerify, LPDWORD lpdwScore);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()	
Id:	DWORD:	: ID number	

Parameters(Output):

lpdwVerify:	LPDWORD	: TRUE	= match
		FALSE	= not match
lpdwScore:	LPDWORD	: 0 ~ 100	= the score of the verification

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.24. Check if there is finger print above sensor

Function Description:

The function detects if a finger is present on the sensor.

Function call:

WORD BioID_IsFinger(LPCOMMDATA lpCommData, BOOL *pbPresent);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()	

Parameters(Output):

pbPresent:	BOOL *	: TURE	// Present
		FALSE	// Not present

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

10.25. Get Max. template number

Function Description:

Get maxi. template store space for module

Function call:

WORD BioID_GetStoredRecordSize (LPCOMMDATA lpCommData, DWORD Id, WORD Index, LPDWORD lpdwSize);

Parameters(Input):

CommData:	LPCOMMDATA	: refer to BioID_Create()
Id:	DWORD	: ID number
Index:	WORD	: Index number

Parameters(Output):

lpdwSize:	LPDWORD	: size

Return code:

WORD:	130 (=CmdErrTime)	Timeout
	7 (=CmdNO)	Error occured during receiving
	6 (=CmdYES)	Success.

11. Get Device ID

In MR650, a 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
 memcpy((PBYTE)deviceID, (PBYTE)pDeviceID + pDeviceID->dwPresetIDOffset,
pDeviceID->dwPresetIDBytes);
}
//////////////////////////////////////////////////////////
```

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

The follow is the API to get UUID in MR650.

```
KernelIoControl(IOCTL_HAL_GET_UUID, NULL, 0, theGUID, sizeof(_GUID), &sizeRead);
```

You can download this sample program about how to get UUID from the link.
http://w3.tw.ute.com/pub/cs/software/sample_program/mr650/MR650_Sample_source.zip

12. Flash Configuration Manager - FlashConfigManager.DLL

The Flash Configuration Manager is a tool to allow third party developers to retain special settings on a Unitech device, which will survive a Cold Boot. The settings are saved in flash memory and accessible via the FlashConfigManager.dll functions described in this document. In general the configuration functions as a way to maintain settings that may or may not be application specific. Functions are provided to check, set, add, and remove individual keys.

12.1. Getting a Configuration Setting

Function Description:

The function to get a current setting is given below:

Function call:

INT GetConfigValue (char * name, void * value);

Parameters(Input):

name: char * : The name of the entry you wish to get

Parameters(Output):

value: void * : A void pointer with no memory allocated to it. The dl will allocate the correct amount of space and copy the appropriate values from the configuration settings.

Return code:

0 upon success. An error code (see "Handling Errors" section) upon failure.

12.2. Updating a Configuration Setting

Function Description:

The function to change the value of an identified setting

Function call:

int SetConfigValue(char * name, void * value);

Parameters(Input):

name: char * : The name of the entry you wish to set.

value: void * : Void pointer to a buffer containing the string to set to. For ease of file system value will be an array of bytes. Value must be already allocated and written. Type restricts the possible values that the array of bytes can be, but does not indicate actual data type. For example for type ‘int’ the appropriate value will be a char * with characters in the ASCII range of 48 to 57 (ie; numbers) not an actual int.

Return code:

0 upon success. error code upon failure.

12.3. Adding a Customer Configuration Setting

Function Description:

This function will create a new entry in the configuration file. Name, type, and value are supplied.

Function call:

INT AddConfigValue (char * name, char * type, void * value);

Parameters(Input):

name: char * : The name of the entry you wish to set.

type: char * : Specifies the type of value this is, which will affect what data is in bounds for the value parameter. Acceptable values and their corresponding value ranges are given below. (bool: "0" or "1", int: only characters 0-9, string: any character)

value: void * : A void pointer with memory allocated to it. The value stored here will be the value returned by GetConfigValue if it is called for the same name.

Return code:

0 upon success. error code upon failure

12.4. Deleting A Configuration Entry

Function Description:

This function will delete any entry you send the name of. It will not be possible to recover the entry in any way.

Function call:

BOOL RmConfigEntry (char *name);

Parameters(Input):

name: char * : Specifiy the name of the configuration you wish to remove. Useful mainly when uninstalling an application as the entry will never be accessible again unless created anew.

Return code:

0 upon success. error code upon failure

12.5. Verifying an Entry's Value

Function Description:

This function provides an easy way to check an entries value against a known possible value.

Function call:

BOOL CheckConfigValue (char *name, void * value);

Parameters(Input):

name: char * : Specify the name of the configuration you wish to check against.

value: void * : enter the value you are looking for.

Return code:

0 upon success. error code upon failure

12.6. Handling Errors

Error Code	Meaning	Possible Causes / Solutions
1	Unable to read current settings	File Settings.conf may be corrupt or missing, replace or Cold Boot
2	Data failed type validation	Data does not fit into boundaries for the given type. If bool use only 0 or 1. If int use only digits 0,1-9
3	Invalid Name Entered	Name doesn't exist for alter / delete or does exist for add. A blank name will also produce this error

13. RS485 communication

13.1. Why RS485?

RS485 is a special implementation of the serial standard. It requires two lines total, with transmit and receive sharing a wire physically. RS485 provides several benefits over standard RS232 communication in terms of the distance data can travel without corruption, and the wide voltage range supported. Since RS485 signaling is differential (difference between the two wires is what defines the signal) it's immunity to electromagnetic interference is relatively high (interference will affect both equally.) This makes RS485 a good choice for many sensor or motor control applications as the distance between the sensor and the control unit may be far and have a high amount of interference. For more information see the standard overview linked below:

http://focus.ti.com/lit/an/slla070c/slla070c.pdf

13.2. Windows APIs Used

Support for RS485 in Windows CE is defined the Winbase.h header file, and is analog to that for RS232 serial port support. The Com port is opened like a data file and can be 'written to' and 'read' with WriteFile and ReadFile. Com port parameters are set with a data structure labeled "DCB" also defined in Winbase.h. For more info see the MSDN page:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/devio/base/dcb_str.asp

13.3. Sample code

Please refer to RS485demo folder from sample program

13.3.1. Opening the Com Port

Function Description:

Com port needs to be opened to grant the application access to its data stream. This is done by creating a virtual file and defining the access permissions for that file. See line 122 of the sample program.

Function call:

BOOL CreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDispostion, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile);

Parameters(Input):

lpFileName: LPCTSTR : Static string defining the name of the file. For the RS485 port use TEXT("COM5:")

dwDesiredAccess: DWORD : Specifies the access your application needs over the opened file. For a com port you need uninhibited read and write access defined as "GENERIC_READ | GENERIC_WRITE"

lpSecurityAttributes: LPSECURITY_ATTRIBUTES : Ignored; set to NULL

dwCreationDispostion: DWORD : How to proceed if the file already exists. For com port (which does already exist) use "OPEN_EXISTING"

dwFlagsAndAttributes: DWORD : Not used, set to 0.

hTemplateFile: HANDLE : Not Used, Set to Null.

Return code:

Function returns a handle for the com port that will be used in all remaining functions.

13.3.2. Setting the Com Port Parameters

Function Description:

Com Port communication has a variety of different possible settings with regards to speed, data correction, flow control, and more. It is necessary to match settings on both ends of the communication for proper correspondence, and for RS485 it is necessary to be able to change the RTS line between sending and receiving data. The best practice here is to gather the default settings into a DCB data structure with GetCommState, then change the key items and reset the port with SetCommTimeouts. See line 131 of the sample program and the Microsoft definition of the DCB struct linked above.

Function call:

void GetCommState(HANDLE hFile, LPDCB lpDCB);

Parameters(Input):

hFile: HANDLE : the Handle to the com port returned by CreateFile.

Parameters(Output):

lpDCB: LPDCB : the DCB structure that the function will write all current settings to.

Function call:

void SetCommState(HANDLE hFile, LPDCB lpDCB);

Parameters(Input):

hFile: HANDLE : the Handle to the com port returned by CreateFile.

lpDCB: LPDCB : the DCB structure that the function will write all current settings to.

For additional control, the port's timeout settings can be altered with SetCommTimeouts and GetCommTimeouts. This is done exactly like the State settings, except using a _COMMTIMEOUTS data struct instead of DCB. See line 137 of the sample program for an example (this will not be necessary for most applications.)

13.3.3. Writing Data To The Com Port

Function Description:

Before writing the port must be set to transmit mode by raising the RTS pin. When that pin is low the port is in receive mode. Direct control of the pin is possible through the Windows API EscapeCommFunction. With the pin high, data can be sent through the port via the WriteFile function. See line 222 of the sample program for an example.

Function call:

BOOL EscapeCommFunction (HANDLE hFile, DWORD dwFunc);

Parameters(Input):

hFile: HANDLE : The handle to the com port.

dwFunc: DWORD : defines what action to take. To raise the RTS line (getting ready to transmit) use SETRTS, to lower it use CLRRTS (after transmitting to be ready to receive.)

Function call:

BOOL WriteFile (HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);

Parameters(Input):

hFile: HANDLE : Handle to the com port to write to.

lpBuffer: LPCVOID : Pointer to buffer containing data to write.

nNumberofBytesToWrite: DWORD : The exact number of bytes you wish to write from the buffer. Often this will be "_tcsclen(lpBuffer)" specific the length of the buffer is to be written.

lpOverlapped: LPOVERLAPPED : Not Supported, set to Null.

Parameters(Output):

lpNumberOfBytesWritten: LPDWORD : pointer to a DWORD that will contain the number of bytes written. Useful to verify that the operation was successful, 0 will indicate a failure.

13.3.4. Reading Data From The Comm Port

Function Description:

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

Function call:

BOOL ReadFile (HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);

Parameters(Input):

hFile:	HANDLE	: Handle to the com port to read from.
nNumberOfBytesToRead:	DWORD	: The exact number of bytes you wish to read from the data stream. The function will pause until this number or a timeout is reached.
lpOverlapped:	LPOVERLAPPED	: Not Supported, set to Null.

Parameters(Output):

lpBuffer:	LPCVOID	: Pointer to buffer containing data to write.
lpNumberOfBytesRead:	LPDWORD	: pointer to a DWORD that will contain the number of bytes read. A value of 0 will indicate there was no data in the stream and timeout occured.

13.3.5. Get error code

Function Description:

Read or Write functions will return Boolean false if there is a failure. The GetLastError() function will then return the reason for the failure.

Function call:

DWORD GetLastError();

Return code:

Returns the error code most recently set. This must be called immediately after the failing function to be relevant. See line 224 of the sample program.

14. Function Key setting on registry

There are 4 function keys on MR650 and they can launch pre-define program when pressed. User can modify registry to define those program.

Below are default registry setting for function

```
;-----------------------------------------
; Registry for Function Key
;-----------------------------------------
[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD\FKEYS]
"FKeyProg10"="Calibration.exe"

;"FKeyProg2"="ctlpnl.exe"
;"FKeyParam2"="Scanner.cpl"
;"FKeyPath2"=""

;"FKeyProg3"="ctlpnl.exe"
;"FKeyParam3"="power"
;"FKeyPath3"=""

;"FKeyProg4"="FUNC9.exe"
;"FKeyParam4"=""
;"FKeyPath4"=""
```

FKeyProg1~4 mean launch program for F1~4 key. FKeyProg10 mean ESC+F1 combination key.
FKeyParam1~4 mean parameter for each program.

15. Update notes

V1.1
- Format define error on Section 10 Get Device ID
- Minor spelling correction.

V1.2
- Camera supporting
- Finger print supporting.

V1.4
- Add camera, finger print and wave sample source code

V1.5
- Add C# sample program link

V1.6
- Add section 5.4 for Back cover status

V1.7
- Add chapter 12 for function key
- RS485 program on chapter 11

V1.8
- Add sample code for get UUID

V1.9
- Add watch dog function

V1.10
- Add Mifare library description

V1.11
- Change logo

V1.12
- Modify download URL

V1.13
- Add the API “Mifare_StartReadEx”, “Mifare_LibraryVersion” in Mifare library description

V1.14
- Add UPEC finger print library description

V1.15
- Modify Mifare library API “Mifare_GetData” to “Mifare_GetDataEx”

V1.16
- Add API description of “Flash Configuration Manager” library