

PA600 Programming Manual

1.	INTRODUCTION	6
1.1	How to download data from scanner	6
1.2	COM definition for PA600	7
2.	USI.DLL – UNITECH SCANNER INTERFACE DLL.....	8
2.1.	Register the application to the USI DLL	8
2.2.	Unregister the application from the USI.DLL	10
2.3.	Enable / Disable Scanner.....	10
2.4.	Reset Scanner	10
2.5.	Get error code	10
2.6.	Returns the system error code	10
2.7.	Get scan data.....	11
2.8.	Get length of scanned data.....	12
2.9.	Get Symbology name	12
2.10.	Clear scan data system buffer	13
2.11.	Good read indicator	13
2.12.	Wait for acknowledgement of the last sent command	13
2.13.	Save setting to profiles	13
2.14.	Save scanner setting into specified file.....	14
2.15.	Change scanner setting from specified setting profile	14
2.16.	Automatically enable scanner beam with pressing trigger key.....	14
2.17.	Stop auto scanning function	14
2.18.	Check if auto scanning is enable	14
2.19.	Check if Scan2Key.exe program is running or not.....	15
2.20.	Test if Scan2Key is enabled.....	15
2.21.	Load/Unload Scan2Key.exe	15
2.22.	Enable/Disable Scan2Key	15
2.23.	Send scanner command to decoding chip.....	16

2.24.	Only send single command decoding chip	16
2.25.	Send command to decoding chip	16
2.26.	Send scanner command set to decoding chip	17
2.27.	Get scanner command set from decoding chip	17
2.28.	Send scanner command set string to decoding chip	17
2.29.	Get scanner command set string from decoding chip	18
2.30.	Get scanner related version information.....	18
2.31.	Enable prompt warning message from USI.....	18
2.32.	Scanner working mode (available for 2D model).....	18
2.33.	Get image (available for 2D model).....	19
2.34.	Resize image (available for 2D model).....	19
2.35.	Save image to file (available for 2D model)	19
2.36.	Get terminator	19
2.37.	Set terminator.....	20
2.38.	Get good read sound mode and sound name	20
2.39.	Set good read sound mode and sound name.....	20
2.40.	Set previewsize (only for 2D engine).....	20
2.41.	Set previewsize time-out (only for 2D engine).....	20
2.42.	Set good read sound mode and sound name.....	21
3.	CONTROL COMMAND FOR DECODER CHIP.....	22
4.	SCANNER3.DLL – BACKWARD COMPATIBLE API FOR PT930/PT930S’S SCANNER3.DLL.....	27
4.1.	Enable Decoder.....	27
4.2.	Disable Decoder.....	27
4.3.	Check barcode input	27
4.4.	Read barcode data	28
4.5.	Get DLL version no	28
4.6.	Reset all symbologies to default.....	28

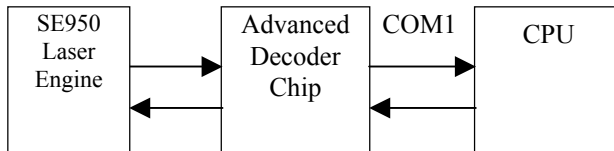
5. SCANKEY3.DLL – BACKWARD COMPATIBLE API FOR PT930/PT930S'S	
SCANKEY3.DLL	29
5.1. Enable Decoder	29
5.2. Disable Decoder	29
5.3. Get DLL version no	29
5.4. Disable laser trigger key	29
5.5. Enable laser trigger key	29
5.6. Reset all symbologies to default	29
6. SYSIOAPI.DLL	30
6.1. Keypad Related Functions	30
6.1.1. Disable/enable power button	30
6.1.2. Set keypad utility input mode	30
6.1.3. Get keypad utility input mode.....	30
6.1.4. Check Alpha key is pressing.....	30
6.1.5. Check Function Key status.....	31
6.1.6. Enable/Disable Function Key.....	31
6.2. Scanner Related Functions	31
6.2.1. Enable/Disable Scanner trigger key	31
6.2.2. Turn on/off Scan Engine.....	31
6.2.3. Get Trigger keys Status	31
6.2.4. Get Scanner Status	32
6.2.5. Check Trigger key is pressing	33
6.3. LED related function	33
6.4. LCD/Backlight related function	34
6.4.1. Screen Backlight Control	34
6.4.2. Get Screen Backlight Status	34
6.4.3. Keypad Backlight Control	34
6.4.4. Get Keypad Backlight Status	34
6.4.5. Screen Backlight Brightness Control	35
6.4.6. Enable/Disable LCD screen.....	35
6.5. SD slot related functions	36
6.5.1. Inquire SD slot status	36
6.5.2. Enable/Disable SD slot	36
6.6. Enable/Disable Vibration	36
6.7. WiFi module related functions	37
6.7.1. Inquire WiFi module status	37
6.7.2. Enable/Disable WiFi module status	37
6.8. Bluetooth related API	38
6.8.1. Enable/Disable Bluetooth Power status	38
6.8.2. Get BT Power Status.....	38

7.	BlueTooth relative API - BTAPI.DLL	39
7.1.	Enable/Disable Bluetooth Power status.....	39
7.2.	Get BT Power Status	39
8.	DYNAMIC LOAD DLL.....	40
9.	USEFUL FUNCTION CALL - WITHOUT INCLUDE SYSIOAPI.DLL.....	41
9.1.	Warm-boot, Cold-boot and power off.....	41
9.2.	GET DEVICE ID	41
10.	UPDATE NOTES	ERROR! BOOKMARK NOT DEFINED.

1. Introduction

1.1 How to download data from scanner

The major difference between the PA600 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 PA600. Inside the PA600 there is an advanced decoding chip to control SE950 laser engine and to handle barcode decoding. Below is system diagram for the PA600 barcode:



According to the above diagram, the PA600 communicates with Decoder Chip by mean of serial port COM4. Its communication parameter is fixed on 38400,N,8.1. Normally, the Decoder Chip is in sleep mode when COM4 is not activated. When COM4 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 PA600.

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 PA600’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, PA600 provides USI.DLL to let user control scanner input, symbologies setting and profile controlling. Please refer to 2 for detail API lists.

USI.DLL is Unitech’s new scanner function library on PA600. For backward compatible issue, Unitech still provide Scanner3.DLL and ScanKey3.DLL for existing PT930/PT930SA user to port their software into PA600, but several APIs on Scanner3.DLL and ScanKey3.DLL have already been removed on PA600. User can refer to 0and 5 for detail supporting API.

1.2 COM definition for PA600

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

2. USI.DLL – Unitech Scanner Interface DLL

Note : For PA600 programming, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide *.H and *.LIB for compiler for mobile 5.0 OS), please refer to Chapter 8 for programming guide.

2.1. Register the application to the USI DLL

Function Description:

Register the application to the USI DLL, so that the DLL can communicate with the application. It will also open and initial scanner port (COM4, 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 GetLastError().

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"="COM4:"
"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();
```

2.3. Enable / Disable Scanner

Function Description:

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

Function call:

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

Parameter: (input)

```
bStatus: TRUE      : Enable Scanner  
        FALSE     : Disable Scanner
```

Return:

```
BOOL : TRUE      : OK  
      FALSE     : Failure
```

2.4. Reset Scanner

Function Description:

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

Function call:

```
BOOL USI_Reset();
```

Return:

Always TRUE

2.5. Get error code

Function Description:

Returns the error code (SERR_***).

Function call:

```
DWORD USI_GetError();
```

Return:

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

2.6. Returns the system error code

Function Description:

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

Function call:

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

Return:

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

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

2.7. Get scan data

Function Description:

Retrieves the scan data into the buffer. USI_GetData returns data in buffer in ASCII code, while USI_GetData2 returns in Unicode. Parameter len specifies the maximum length of the buffer. Returns the length of characters. It also returns the barcode type if type is not NULL. Return 0 means that the buffer is too short to hold the data.

The scanner will be disabled when scanner successfully decoded a data, so the data is protected from been destroyed by trying continuously scanning. Call USI_GetData / USI_GetData2 or USI_ResetData will automatically enable the scanner.

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 and enable scanner, 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.

For a complete list of barcode types, refer to the header file USI.H for the predefined data with BCT_ prefix

Function call:

```
UINT USI_GetData(LPBYTE buffer, UINT len, UINT* type);
UINT USI_GetData2(LPTSTR 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

Return:
 UINT : Data length

2.8. Get length of scanned data

Function Description:

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

Function call:

UINT USI_GetDataLength();

Return:

UNIT : Data length

2.9. Get Symbology name

Function Description:

Returns the barcode name of the type

Function call:

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

Parameter: (input)

type: UINT : barcode type. (refer to below 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:

BOOL : TRUE : OK
FALSE : Failure

2.10. Clear scan data system buffer

Function Description:

Enable scanner and reset the data buffer so that next new scan data can come in.

Function call:

void USI_ResetData();

2.11. Good read indicator

Function Description:

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

Function call:

void USI_ReadOK();

Note:

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

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

The function prototype of GoodReadLEDOn is:

```
VOID WINAPI GoodReadLEDOn(BOOL fon);
```

Turn on when fon is TRUE, and turn off when fon is FALSE.

2.12. Wait for acknowledgement of the last sent command

Function Description:

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

Function call:

BOOL USI_WaitForSendEchoTO(DWORD timeout);

Parameter: (input)

timeout: DWORD : Specifies the timeout in millisecond.

Return:

Returns FALSE if timeout.

2.13. Save setting to profiles

Function Description:

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

Function call:

BOOL USI_SaveCurrentSettings();

Return:

TRUE 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:

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:

TRUE = success

FALSE = error

2.16. Automatically enable scanner beam with pressing trigger key

Function Description:

Start auto scanning. Scan engine will be automatically triggered 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:

BOOL: TRUE : auto-scanning is running

FALSE : auto-scanning is disabled.

2.19. Check if Scan2Key.exe program is running or not

Function Description:

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

Function call:

HWND S2K_IsLoaded();

Return:

NULL: Scan2Key is not running

Non-NULL: indicates scan2key is running. It actually returns window handle for scan2key, but it is for internal use – send messages.

2.20. Test if Scan2Key is enabled

Function Description:

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

Function call:

BOOL S2K_IsEnabled();

Return:

TRUE = enabled.

FALSE = 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:

TRUE = successfully loaded.

2.22. Enable/Disable Scan2Key

Function Description:

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

Function call:

BOOL S2K_Enable(BOOL enable, DWORD timeout);

Parameter: (input)

enable: BOOL : TRUE = Enable scanned data to keyboard buffer
FALSE = Disable scanned data to keyboard

timeout: DWORD : when enable or disable Scan2Key, it will wait until Scan2Key has been removed from memory or timeout specified by this parameter.

Return:

TRUE = if successfully enabled Scan2Key, otherwise FALSE

2.23. Send scanner command to decoding chip

Function Description:

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

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

Please refer to complete command reference on section 4

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

Parameter: (input)

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

Return:

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

2.24. Only send single command decoding chip

Function Description:

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

Esc, 0x80+cmd

Function call:

BOOL HAM_SendCommand1(BYTE cmd);

Parameter: (input)

cmd: BYTE : command

Return:

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

2.25. Send command to decoding chip

Function Description:

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

Esc, parameter1, parameter2, ..., BCC

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

Function call:

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

Parameter: (input)

num: BYTE : number of total parameters
parameterx: BYTE : Parameter

Return:

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

2.26. Send scanner command set to decoding chip

Function Description:

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

To send a string please call HAM_SendCommand_SetString.

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

Parameter: (input)

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

Return:

TRUE = indicates the setting has been set successfully.

2.27. Get scanner command set from decoding chip

Function Description:

This function call has the similar function as HAM_SendCommand_Set except that it retrieves setting from scanner. This is a synchronized function, it returns when command has been sent to and got response from scanner. USI_WaitForSendEchoTO is not needed before a next continuous send command call.

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

Parameter: (input)

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

Return:

TRUE = indicates the setting has been retrieve successfully.

2.28. Send scanner command set string to decoding chip

Function Description:

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

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

Parameter: (input)

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

Return:

TRUE = indicates the setting has been set successfully.

2.29. Get scanner command set string from decoding chip

Function Description:

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

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

Parameter: (input)

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

Return:

TRUE = indicates the setting has been retrieve successfully.

2.30. Get scanner related version information

Function Description:

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

Function call:

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

Parameter: (output)

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

Return:

Always True.

2.31. Enable prompt warning message from USI

Function Description:

Enables USI to report working information in a popup window

Function call:

BOOL USI_EnablePromptMessage(BOOL enable);

Parameter: (output)

enable: BOOL : True= enable, Fail:Disable

Return:

Always True.

2.32. Scanner working mode (available for 2D model)

Function Description:

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

Function call:

BOOL USI_SetWorkingMode(int mode);

Parameter: (output)

mode: int : mode = SWM_BARCODE - Barcode
mode = SWM_IMAGE - image capture
mode = SWM_IMAGE_PREVIEW - preview and image capture

for 2D scanner

Return:

Always True.

2.33. Get image (available for 2D model)

Function Description:

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

Function call:

HBITMAP USI_GetImageBitmap(SIZE* imagesize);

Parameter: (output)

imagesize: SIZE : Bitmap image size

Return:

HBITMAP : image

2.34. Resize image (available for 2D model)

Function Description:

Resizes a bitmap.

Function call:

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

Parameter: (input)

hbmp: HBITMAP : the bitmap handle needs to be resized.

cx: int: : cx define new width dimension of the bmp.

cy int: : cy define new height dimension of the bmp.

kepratio: BOOL: : If kepratio is true, it will resize the bmp to fit the cx-cy rectangle and keep the ratio of the original image.

Return:

HBITMAP : image

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

Function Description:

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

Function call:

BOOL USI_SaveImageToFile(LPCTSTR filename, int compressionfactor);

Parameter: (input)

filename: LPCTSTR : File extension name defines format. HHP, SSI support .bmp, .jpg, and .tif. Tohken supports .bmp and .jpg.

compressionfactor: int : compressionfactor is used for when format is jpg.

Return:

Always True.

2.36. Get terminator

Function Description:

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

Function call:

int USI_GetTerminator();

Return:

Int : terminator

2.37. Set terminator

Function Description:

Sets terminator.

Function call:

BOOL USI_SetTerminator(int terminator);

Parameter: (input)

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

Return:

Always True.

2.38. Get good read sound mode and sound name

Function Description:

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

Function call:

int USI_GetGoodReadEcho(LPTSTR buffer, UINT blen);

Parameter: (input)

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

Return:

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

2.39. Set good read sound mode and sound name

Function Description:

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

Function call:

BOOL USI_SetGoodReadEcho(int mode, LPTSTR SoundFileName);

Parameter: (input)

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

Return:

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

2.40. Set preview size (only for 2D engine)

Function Description:

Defines how large is the window for image preview.

Function call:

void USI_SetPreviewSize (SIZE size);

Parameter: (input)

size:	SIZE	:	Size of image preview window
-------	------	---	------------------------------

2.41. Set preview size time-out (only for 2D engine)

Function Description:

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

Function call:

void USI_SetPreviewTimeout (DWORD timeout);

Parameter: (input)

timeout:	DWORD	:	timeout in seconds
----------	-------	---	--------------------

2.42. Set good read sound mode and sound name

Function Description:

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

Function call:

BOOL USI_SetGoodReadEcho(int mode, LPTSTR SoundFileName);

Parameter: (input)

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

Return:

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

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 "Jc1"
Mterminator	Esc,0,4,31,m,S1,BCC	Here S1 is 0_ENTER,1 NONE

	No meaning for you	
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¬ 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)
l2of5_set	Esc,0,4,40,m,S1,BCC	S1 is S1.B0 is for l2of5_enable,S1.B1 is for l2of5_fixlength,S1.B3B2 is for l2of5_cd,S1.B5B4 is for l2of5_ss
l2of5_enable	Esc,0,4,41,m,S1,BCC	S1 is =0 disable,1_enable
l2of5_fixlength	Esc,0,4,42,m,S1,BCC	S1 is =0 on,1_off (record first 3 record len)
l2of5_cd	Esc,0,4,43,m,S1,BCC	S1 is =0 calculate&send,1 calculate¬ send,2_no calculation
l2of5_ss	Esc,0,4,44,m,S1,BCC	S1 is 0 first digit suppressed,1 last digit suppressed,2_not supressed
l25_min	Esc,0,4,45,m,S1,BCC	S1 is 2~64:10 (min<=data len)
l25_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¬ 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,Li,91,m,S1...Si,BC C	Lh/Li 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,Li,a3,S1,S2,...,Sn, BCC	<ul style="list-style-type: none"> 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,Li,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,Li,string					
Here: The Lh/Li is string length, Lh is high byte, Li is low byte, The string length is excluded the Code_number and Lh/Li. 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. **Scanner3.DLL – Backward compatible API for PT930/PT930S's Scanner3.dll**

Note : For PA600, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide *.H and *.LIB for compiler), please refer to Chapter 8 for programming guide.

4.1. **Enable Decoder**

Function Description: This function will open COM4 port, create a thread to get any barcode input from Decoder Chip, and then store input data in the system buffer. Application can use function call **PT_GetBarcode()** to get input data from the system buffer.

Function call:

```
INT PT_EnableBarcode(VOID);
```

Return code:

```
=1      Create new thread fail
=2      Cannot re-enable
=3      Cannot open COM4
=4      Upload parameter from Hamster fail
=0      OK
```

4.2. **Disable Decoder**

Function Description:

This function will close COM4 port and then remove thread which is created by **PT_EnableBarcode()**

Function call:

```
VOID PT_DisableBarcode( VOID );
```

4.3. **Check barcode input**

Function Description:

This function is used to check whether there is available barcode data on system buffer which is successfully decoded by decoder chip.

Function call:

```
BOOL PT_CheckBarcode( VOID );
```

Return code:

```
TRUE = There is input data on system buffer.
FALSE = There is no data on system buffer.
```

4.4. Read barcode data

Function Description: Get input barcode data and its type from system buffer.

Function call: BOOL PT_GetBarcode(TCHAR *szBarcodeBuffer,TCHAR *cType);

Parameter: (output)

szBarcodeBuffer : string buffer for storing input data

cType : Type of Input data

=00H Full Code 39
=01H STD Code 39
=02H EAN-13
=03H UPC-A
=04H EAN-8
=05H UPC-E
=06H I-25
=07H CODABAR
=08H Code 128
=09H Code 93
=0Ah STD 25
=0BH MSI
=0CH EAN-128
=0DH Code 32
=0EH DELTA
=0FH LABEL
=10H PLESSEY
=11H Code 11
=12H TOSHIBA

Return code: TRUE = There is barcode input

FALSE = No Barcode Input

4.5. Get DLL version no

Function description:

This function is used to get DLL version no.

Function call:

INT PT_DllVersion(void);

Return :

Integer

4.6. Reset all symbologies to default

Function Description:

This function call will reset decoder chip's symbologies setting to system default value

Function call for VC:

int PT_SetToDefault (VOID)

Function call for VB:

PT_SetToDefault

5. **ScanKey3.DLL – Backward compatible API for PT930/PT930S’s ScanKey3.dll**

Note : For PA600, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide “USI.H” and “USI.LIB” for compiler), please refer to Chapter 8 for programming guide.

5.1. **Enable Decoder**

Function Description: This function will open COM4 port, create a thread to get any barcode input from Decoder Chip, and then send scanner data to keyboard buffer. User application can get input data just like standard keyboard input.

Function call for VC: int PT_EnableBarToKey(VOID)

Return code:

=1	Create new thread fail
=2	Can not re-enable
=3	Can not open COM4
=4	Upload parameter from Hamster fail
=0	OK

5.2. **Disable Decoder**

Function Description: This function will close COM4 port and then remove thread which is created by **PT_EnableBarToKey()**

Function call for VC: VOID PT_DisableBarToKey (VOID)

5.3. **Get DLL version no**

Function description: This function is used to get DLL version number.

Function call for VC: INT PT_Version(void);

Return : Integer

5.4. **Disable laser trigger key**

Function Description:

This function only stop trigger key to activate laser beam, so COM4 port is still open. This function call is useful when some fields is only allow keyboard input..

Function call for VC:

int PT_StopScan (VOID)

5.5. **Enable laser trigger key**

Function Description: This function only stop trigger key to activate laser beam, so COM4 port is still open. This function call is useful when some fields is only allow keyboard input..

Function call for VC: int PT_StartScan (VOID)

5.6. **Reset all symbologies to default**

Function Description: This function call will reset decoder chip's symbologies setting to system default value

Function call for VC: int PT_SetToDefault (VOID)

Function call for VB: PT_SetToDefault

6. SysIOAPI.DLL

This DLL provide hardware relative API for user to control scanner, LED, back-light and PC card slot. API functions are provided through DLL to assist programmer to write application for PA600.

Note : For PA600 programming, it need to dynamically load DLL for using Unitech built-in DLL (Unitech will not provide *.H and *.LIB for compiler), please refer to Chapter 8 for programming guide.

6.1. Keypad Related Functions

6.1.1. Disable/enable power button

Function Description:

To enable / disable power button

Function call:

VOID DisablePowerButton (BOOL)

Parameter (Input)

TRUE = Disable power button.

FALSE = Enable power button.

Return code:

None

6.1.2. Set keypad utility input mode

Function Description:

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

Function call:

void SetGetVKWorkingMode(int)

Parameter (input)

0 = hide the selection window.

1 = show lower case selection window.

2 = show upper case selection window.

Return code:

None

6.1.3. Get keypad utility input mode

Function Description:

This function is used to check alpha key input mode.

Function call:

BYTE GetAlphaKeyWorkingMode(void);

Return code:

0 = **normal** .

1 = **lower case**.

2 = **upper case**.

6.1.4. Check Alpha key is pressing

Function Description:

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

Function call:

BOOL GetKeypadAlphaKeyStatus(void);

Return code:

TRUE = Alpha key is pressed.

FALSE = Alpha key is released.

6.1.5. Check Function Key status

Function Description:

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

Function call:

BOOL GetFnKeyStatus(void)

Return code:

TRUE = Function key is enabled.
FALSE = Function key is disabled.

6.1.6. Enable/Disable Function Key

Function Description:

This function is used to enable/disable function key.

Function call:

void SetFnKeyDisable(BOOL bOff)

Parameter (Input):

TRUE = Disable function key.
FALSE = Enable function key.

6.2. Scanner Related Functions

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

6.2.1. Enable/Disable Scanner trigger key

Function Description:

This function enables/disables trigger keys.

Function call:

void EnableScannerTrigger(BOOL fOn)

Parameter (Input)

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

Return code:

6.2.2. Turn on/off Scan Engine

Function Description:

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

Function call:

void SetScannerOn(BOOL fON)

Parameter(Input)

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

Return code: none

6.2.3. Get Trigger keys Status

Function Description:

This function returns enable/disable status of trigger keys.

Function call:

BOOL GetScannerTrigger(void)

Return code:

TRUE = trigger keys are enabled.
FALSE = trigger keys are disabled.

6.2.4. Get Scanner Status

Function Description:

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

Function call:

BOOL GetScannerStatus(void)

Return code:

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

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

6.2.5. Check Trigger key is pressing

Function Description:

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

Function call:

```
BOOL TriggerKeyStatus( int key);
```

Parameter(Input)

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

Return code:

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

Example:

```
#define kKeybdTriggerEventName          TEXT("KeybdTriggerChangeEvent")  
#define kKeybdAlphaKeyName             TEXT("KBDAlphaKeyChangeEvent")  
#define LEFT_TRIGGER_KEY 1  
#define RIGHT_TRIGGER_KEY 2  
gKeyEvents[0] = CreateEvent(NULL, TRUE, FALSE, kKeybdTriggerEventName);  
gKeyEvents[1] = CreateEvent(NULL, TRUE, FALSE, kKeybdAlphaKeyName);  
  
while (1)  
{  
    WaitForMultipleObjects(2, gKeyEvents, FALSE, INFINITE);  
  
    TriggerKeyStatus(LEFT_TRIGGER_KEY);  
    TriggerKeyStatus(RIGHT_TRIGGER_KEY);  
}
```

6.3. LED related function

Function Description:

There are two LEDs above the screen of PA600, red and green LEDs. Only the green LED can be controlled by programmer.

Function call:

```
void GoodReadLEDOn(BOOL fON)
```

Parameter(Input)

```
fON:  BOOL:      TRUE = turn on LED.  
      FALSE = turn off green LED.
```

6.4. LCD/Backlight related function

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

6.4.1. Screen Backlight Control

Function Description:

This function turns screen backlight on or off.

Function call:

void BacklightOn(BOOL fON)

Parameter(Input)

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

Return code:

6.4.2. Get Screen Backlight Status

Function Description:

This function returns the status of screen backlight.

Function call:

BOOL GetBacklightStatus(void)

Return code:

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

6.4.3. Keypad Backlight Control

Function Description:

This function turns keyoad backlight on or off.

Function call:

void KeypadLightOn(BOOL fON)

Parameter(Input)

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

Return code:

6.4.4. Get Keypad Backlight Status

Function Description:

This function returns the status of keyoad backlight.

Function call:

BOOL GetKeypadLightStatus(void)

Return code:

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

6.4.5. Screen Backlight Brightness Control

Function Description:

This function adjusts screen backlight brightness.

Function call:

void BrightnessUp(BOOL fup)

Parameters(Input)

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

Return code:

6.4.6. Enable/Disable LCD screen

Function Description:

Turn on / off LCD screen

Function call:

void PowerOnColorLCD(BOOL fON)

Parameters(Input)

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

Return None

6.5. SD slot related functions

6.5.1. Inquire SD slot status

Function Description:

This function returns SD slot enable/disable status.

Function call:

BOOL GetSDStatus()

Return

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

6.5.2. Enable/Disable SD slot

Function Description:

This function enables/disables SD slot.

Function call:

void EnableSDSlot(BOOL bEnable)

Parameters(Input)

bEnable : BOOL: TRUE = enable slot.
FALSE = disable slot.

6.6. Enable/Disable Vibration

Function Description:

This function enables/disables Vibrator.

Function call:

void VibrationOn(BOOL fOn)

Parameters(Input)

fOn: BOOL: TRUE = ON.
FALSE = OFF

6.7. WiFi module related functions

6.7.1. Inquire WiFi module status

Function Description:

This function returns WiFi module enable/disable status.

Function call:

BOOL GetWLANStatus()

Return

BOOL: TRUE = module is enabled.
FALSE = module is disable.

6.7.2. Enable/Disable WiFi module status

Function Description:

This function enables/disables WiFi module.

Function call:

void WLANPowerEnable(BOOL fOn)

Parameters(Input)

fOn: BOOL: TRUE = enable module.
FALSE = disable module.

6.8. Bluetooth related API

Note : Not like Unitech's PA9xx series, Bluetooth related APIs are implemented on SysIOAPI.DLL. It is suggested to use those API from SysIOAPI. But, we still keep the same function call on BTAPI.DLL for backward compatible issue.

6.8.1. Enable/Disable Bluetooth Power status

Function Description:

Enable Bluetooth Module Power ON/OFF

Function call:

void BT_PowerEnable (BOOL bEnable)

Parameter (Input)

bON: BOOL: TRUE = Enable

FALSE = Disable

Return code:

None

6.8.2. Get BT Power Status

Function Description:

Get Bluetooth Module Power Status

Function call:

BYTE BT_PowerStatus (void)

Return code:

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

7. BlueTooth relative API - BTAPI.DLL

This DLL provide BlueTooth relative API. Two files are essential and provided in SDK, BTAPI.LIB and BTAPI.H.

7.1. Enable/Disable Bluetooth Power status

Function Description:

Enable Bluetooth Module Power ON/OFF

Function call:

void BT_PowerEnable (BOOL bEnable)

Parameter (Input)

bON: BOOL: TRUE = Enable

FALSE = Disable

Return code:

NONE

7.2. Get BT Power Status

Function Description:

Get Bluetooth Module Power Status

Function call:

BYTE BT_PowerStatus (void)

Return code:

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

8. **Dynamic Load DLL**

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

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

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

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

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

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

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

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

////////////////////////////////////
```


9. Useful function call - without include SysIOAPI.DLL

9.1. 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 );
}
```

9.2. **Get Device ID**

In PA600, 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], szProductID[100], stringBuffer[100];
BYTE szBuff[255];
DWORD bytesReturned;
char platformID[64];

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

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

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

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