

DTR SDK User Guide

For DT4XX WinCE 6.0

Version 1.0

Embedded Development Group

Revision History

Date	Version	Author	Description
2010-01-11	V0.1	Zenghua Gao	How to use the SDK for DTOSK.
2010-01-18	V0.1	Lei Sun	Enable and disable the devices including WiFi, Bluetooth, RFID, MSR and Scanner
2010-2-26	V0.1	Jeimei Chen	Add DTRControlSuspend.dll to OS and DTRControlSuspend.h to SDK. This DLL exports the API to set OS suspend.
2010-2-26	V0.1	Junbo Yin	Add ButtonManager SDK.
2010-3-3	V0.1	Junbo Yin	Add Scanner, MSR, and RFID SDK.
2010-3-16	V0.1	Junbo Yin	Support RTC resume

Table of Contents

DTR SDK User Guide	1
Chapter 1 DTOSK SDK	4
1.1 Background	4
1.2 Function Descriptions	4
1.3 DTOSK SDK Demo Program	4
Chapter 2 Device Control SDK	5
3.1 Background	5
3.2 Function Description	5
3.3 Device Control SDK Demo Program	6
Chapter 3 Set Suspend SDK	7
3.1 Background	7
3.2 Function Descriptions	7
3.3 Set Suspend SDK Demo Program	7
Chapter 4 Button Manager SDK	8
4.1 Background	8
4.2 Function Description	8
5.2.1 int GetFreeButtons(BYTE* pFreeBtnBuf, UINT* pFreeBtnNum, int flag)	8
5.2.2 BOOL OccupyButtons(BYTE* pFreeBtnBuf, UINT bufLen, int flag)	8
5.2.3 BOOL ReleaseButtons(BYTE* pOccupiedBtns, UINT bufLen, int flag)	9
5.2.4 4.2.4 BOOL SetBtnEventCallback(ButtonEventCallback BtnEventFun)	9
4.3 Demo	10
Chapter 5 Scanner SDK	11
5.1 Background	11
5.2 Function	11
5.2.1 unsigned short unsigned short Scanner_Connect(int port)	11
5.2.2 unsigned short Scanner_Disconnect (void)	11
5.2.3 unsigned short Scanner_GetFirmwareInfo(char *version, long verLen)	11
5.2.4 unsigned short Scanner_GetTriggerMode(int *mode)	12
5.2.5 unsigned short Scanner_SetTriggerMode(int mode)	12

5.2.6	unsigned short Scanner_SetDataReceiverCallback (ScannerDataCallback revFun)	13
5.2.7	unsigned short Scanner_SetFactoryDefault()	13
5.2.8	unsigned short Scanner_SetBeepStatus(BOOL bBeep)	14
5.2.9	unsigned short Scanner_GetType(long *type)	14
5.2.10	unsigned short Scanner_SetSymbolIdentifierStatus(int index, BOOL bActive)	14
5.2.11	unsigned short Scanner_GetSymbolIdentifierStatus(int index, BOOL* bActive)	15
5.2.12	unsigned short Scanner_EnumSymbolIdentifierClass(IscpRsSymbologies* pSymbologies, DWORD* count)	15
5.2.13	unsigned short Scanner_Trigger(void)	16
5.2.14	unsigned short Scanner_UnTrigger (void)	16
Chapter 6 MSR SDK		17
6.1	Background	17
6.2	Function	17
6.2.1	unsigned short Msr_Connect(int port)	17
6.2.2	unsigned short Msr_Disconnect (void)	17
6.2.3	unsigned short GetFirmwareInfo(char *name, long nameBufLen, char *version, long verBufLen)	17
6.2.4	unsigned short Msr_GetPortParameters(int *baud, int * databits, int *parity, int* stopbits)	18
6.2.5	unsigned short Msr_SetPortParameters(int baud, int databits, int parity, int stopbits)	18
6.2.6	unsigned short Msr_SetDataReceiverCallback (MsrDataCallback revFun)	19
6.2.7	unsigned short Msr_ParseEnable (BOOL bParse, struct PAESEDATAFORMAT *parseFormat, int parseCount)	20
Chapter 7 FRID SDK		21
7.1	Background	21
7.2	Function	21
7.2.1	int RFID_Connect(int port)	21
7.2.2	int RFID_Disconnect(void)	21
1.	0: Failed to disconnect. This may be due to that the connection is not exist, or RFID reader is busy. RFID_SetUidReceiverCallback	21
7.2.3	int RFID_SetUidReceiverCallback(RFIDUidCallback pUidFun)	21
7.2.4	int RFID_ReadBlockData (const char* pszUid, const char* pszPassword, unsigned int nBlkIdx, unsigned char* pBuf, unsigned int* pBufSize)	22
7.2.5	int RFID_GetFirmwareInfo(char* pVerBuf, int nBufSize)	22
Chapter 8 RTC SDK		24
8.1	Background	24
8.2	Function	24
8.2.1	int RFID_Connect(int port)	24

Chapter 1 DTOSK SDK

1.1 Background

DTOSK has three different keyboard layouts; these are the alphabet layout, the numeric layout and the symbol layout. As for customers' request, this SDK provides customers a method to choose which layout of DTOSK should be shown next time when DTOSK is launched.

1.2 Function Descriptions

void ChangeDTOSKUIState(DWORD SelectNum);

Change the DTOSK layout shown on the next launch time.

Parameters:

DWORD SelectNum [in]

This parameter defines the keyboard layout of choosing. The macros DTR_DTOSK_ALPHABET, DTR_DTOSK_NUMBER and DTR_DTOSK_SYMBOL as defined in the header file DTDLL.h should be used for this parameter.

Return Values: NONE.

SDK files: DTDLL.h, DTDLL.lib and DTDLL.dll.

1.3 DTOSK SDK Demo Program



The utility DTRTest is for both DTOSK SDK and DTR Device Control SDK demo purpose. Select a keyboard layout in the DTOSK DEMO list control and click the button Change, and then launch the DTOSK again to see the change.

Chapter 2 Device Control SDK

3.1 Background

There are several peripheral devices as defined in DTR hand-held. These are the WiFi device, the Bluetooth device, the MSR device, the RFID device and the scanner. This SDK provides APIs for disabling and enabling the devices.

3.2 Function Description

BOOL DTEnableDevice(DWORD DeviceID);

Call this function to enable the device designated by the parameter *DeviceID*.

Parameters:

DWORD DeviceID [in]

This parameter is used to designate the device to be disabled or enabled. The macros `DTR_WIFI_DEVICE`, `DTR_BLUETOOTH_DEVICE`, `DTR_RFID_DEVICE`, `DTR_MSR_DEVICE`, `DTR_SCANNER_DEVICE` are defined in the header file `DTDLL.h` for each device.

Return Values:

Return 1 if the device exists and is enabled successfully;

Return 0 if it's failed to enable the designated device;

Return 2 if device does not exist

BOOL DTDisableDevice(DWORD DeviceID);

Call this function to disable the device designated by the parameter *DeviceID*.

Parameters:

DWORD DeviceID [in]

This parameter is used to designate the device to be disabled or enabled. The macros `DTR_WIFI_DEVICE`, `DTR_BLUETOOTH_DEVICE`, `DTR_RFID_DEVICE`, `DTR_MSR_DEVICE`, `DTR_SCANNER_DEVICE` are defined in the header file `DTDLL.h` for each device.

Return Values:

Return 1 if the device is disabled successfully, or 0 for failure.

3.3 Device Control SDK Demo Program



The utility DTRTest is for both DTOSK SDK and DTR Device Control SDK demo purpose. Select a device macro in the DEVICE CONTROL DEMO list control and click the button Enable or Disable to enable or disable the device respectively.

Chapter 3 Set Suspend SDK

3.1 Background

DTR hand-held devices support full suspend mode and WiFi alive suspend mode. With the WiFi alive suspend mode, the device enters the suspend mode while keeping the WiFi connection alive. This SDK provides the customers an API to enter the preferred suspend mode.

3.2 Function Descriptions

HRESULT DTRControlSuspend(*bool bfullsuspend*)

This function is used to set OS to enter full suspend or WiFi alive suspend mode.

Parameters

Bool bfullsuspend [in] true for full suspend and false for wifi alive suspend status

Return Values

S_OK: The function is executed successfully.

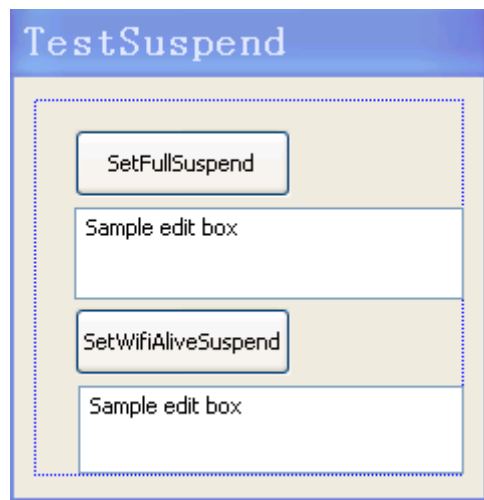
E_ERROR_SETWIFI: Failed to enter the full suspend mode. It indicates that the WiFi device is setting to On or Off at the same time of the function running.

E_ERROR_WAIT: Failed to enter the full suspend mode. The interval from previous System Resume to the next full suspend must >12sec.

E_FAIL: Failed to set.

3.3 Set Suspend SDK Demo Program

The utility TestSuspend is a demo program of the Set Suspend SDK. It has a straightforward GUI as below.



Chapter 4 Button Manager SDK

4.1 Background

As for the customer requests, DTR Button Manager SDK provides the customer APs the mechanism to co-work with DTR Button Manager. The presumption is that only those buttons which are not occupied by Button Manager could be used by the SDK applications. The Button Manager SDK provides APIs to APs, therefore the APs can get the free button event and assign function to that button.

4.2 Function Description

5.2.1 **int GetFreeButtons(BYTE* pFreeBtnBuf, UINT* pFreeBtnNum, int flag)**

Call this function to get the total number of the free buttons and the indices of every free button.

Parameters

pFreeBtnBuf: [OUT] Specify the index of free buttons.

pFreeBtnNum:[IN/OUT] When in, it is the size of pFreeBtnBuf; When out, it is the real number copy to pFreeBtnBuf.

flag:[IN] It is the button's environment. The environment includes "Normal", "Fn ON" which is alias of compose key. 0 for "Normal", 1 for "Fn ON".

Return Values

0: There is not any more free buttons.

>0: Specify the total number of free buttons.

Remark

If the value of *pFreeBtnNum is less than return value, which means the free button buffer size is too small, Should enlarge the buffer size.

5.2.2 **BOOL OccupyButtons(BYTE* pFreeBtnBuf, UINT bufLen, int flag)**

This function is used to occupy some free buttons which are from GetFreeButtons function. Once successfully occupy specified free buttons, it will receive the button event (pressed).

Parameters

pFreeBtnBuf:[IN] Specify the index of free buttons.

bufLen:[IN] It is the size of pFreeBtnBuf.

flag:[IN] It is the button's environment. The environment includes "Normal", and "Fn ON". 0 for "Normal", 1 for "Fn ON".

Return Values

- 1: Successfully occupy all specified free buttons.
- 0: Any one specified button is failed to occupy. Users should call GetFreeButtons function again to get free buttons.

Remark

If a free button is occupied, it will be marked with “Occupied” until release. So remember release the occupied buttons when complete.

5.2.3 BOOL ReleaseButtons(BYTE* pOccupiedBtns, UINT bufLen, int flag)

This function is used to release the occupied buttons.

Parameters

pOccupiedBtnBuf:[IN] Specify the index of occupied buttons.

bufLen:[IN] It is the size of pOccupiedBtnBuf.

flag:[IN] It is the button’s environment. The environment includes “Normal”, and “Fn ON”. 0 for “Normal”, 1 for “Fn ON”.

Return Values

- 1: Successfully release all specified occupied buttons.
- 0: Any of specified buttons to be release is not occupied.

Remark

If does not release the occupied buttons, others program can not get a chance to occupy it.

5.2.4 4.2.4 BOOL SetBtnEventCallback(ButtonEventCallback BtnEventFun)

This function is used to specify a button event receiver function. When button event occurs, this function will be called.

Parameters

BtnEventFun:[IN] a function pointer.

Return Values

- 1: Successfully release all specified occupied buttons.
- 0: Any of specified buttons to be release is not occupied.

Remark

The callback function is defined as following.

```
typedef void (__stdcall*ButtonEventCallback)(BYTE btnIndex, int btnStatus, int flag).
```

“btnIndex” is button index which can distinguish which button’ event. “btnStatus “ describes the event status (0 for released, 1 for pressed). “Flag” describes the button environment, which will be “Normal”, “Fn ON”.

4.3 Demo

DTBtnEventDemo.exe uses the SDK to test APIs.



“Get”, “Occupy” and “Release” correspond with “GetFreeButtons” ,” OccupyButtons”, “ReleaseButtons” APIs.

Chapter 5 Scanner SDK

5.1 Background

Provide a DLL to operation Scanner device. DLL file name is DTScannerAPI.dll. LIB file name is DTScannerAPI.lib. Header file name is DTScannerAPI.h. The Scanner port parameters are saved in registry:

HKEY_LOCAL_MACHINE\SOFTWARE\DT Research\ScannerConfig

5.2Function

5.2.1 unsigned short unsigned short Scanner_Connect(int port)

This function is used to establish host connection with the scanner.

Parameters

port: [IN] the number of serial port used to connect to Scanner, reserved 1.

Return Values

0: Succeed.

1: Interface port not available

2: No device found

5.2.2 unsigned short Scanner_Disconnect (void)

This function is used to close the host connection with the scanner.

Parameters

None

Return Values

0: Succeed.

3: Timeout

4: Scanner had not been connected

5.2.3 unsigned short Scanner_GetFirmwareInfo(char *version, long verLen)

This function is used to get the scanner firmware information.

Parameters

version:[OUT] pointer to a buffer that will be filled in firmware version.

verLen: [IN] version buffer length.

Return Values

0: Succeed.

3: Timeout

4: Scanner had not been connected

10: Invalid input buffer.

11: Input buffer length too small

Remark

For code example:

```
char version[256];
```

```
unsigned short ret=0;
```

```
ret = Scanner_GetFirmwareInfo(version,256);
```

5.2.4 unsigned short Scanner_GetTriggerMode(int *mode)

This function is used to get current trigger mode scanner used.

Parameters

mode: [OUT] pointer to trigger mode index, which will be filled in the current trigger mode index.

Return Values

0: Succeed.

3: Timeout

4: Scanner had not been connected

6: Not supported mode

Remark

The trigger mode predefined:

Index	0	1	2	3	4	5	6
Trigger Mode	continuous	level	pulse	flashing	autostand	toggle	presentation

5.2.5 unsigned short Scanner_SetTriggerMode(int mode)

This function is used to set scanner trigger mode to which mode.

Parameters

mode: [IN] the trigger mode index.

Return Values

0: Succeed.

3: Timeout

4: Scanner had not been connected

6: Not supported mode

5.2.6 unsigned short Scanner_SetDataReceiverCallback (ScannerDataCallback revFun)

This function is used to identify the host application a data receive callback function, when data received from Scanner, that function will be called.

Parameters

revFun:[IN] a function pointer.

Return Values

0: Succeed.

1: Function pointer is invalid

Remark

The callback function is defined as following.

```
typedef void (__stdcall *ScannerDataCallback)(void *pBuf, long dataLen)
```

pBuf parameter is a pointer which point to Scanner data buffer.

dataLen parameter will give data length in buffer.

For application code example.

```
void ReceiveScannerData(void *pBuf, long dataLen);
```

```
//.....
```

```
Scanner_SetDataReceiverCallback(&ReceiveScannerData);
```

5.2.7 unsigned short Scanner_SetFactoryDefault()

This function is used to make scanner to its default setting.

Parameters

None

Return Values

- 0: Succeed.
- 3: Timeout
- 4: Scanner had not been connected.

5.2.8 unsigned short Scanner_SetBeepStatus(BOOL bBeep)

This function is used to enable or disable beep function when received chars.

Parameters

bBeep:[IN] FALSE means to disable beep, TRUE means to enable beep.

Return Values

- 0: Succeed.
- 3: Timeout
- 4: Scanner had not been connected.

5.2.9 unsigned short Scanner_GetType(long *type)

This function is used to get the Scanner's type, such as 1D or 2D.

Parameters

type:[OUT] To identify the scanner's type. 1 means 1D, 2 means 2D.

Return Values

- 0: Succeed.
- 3: Timeout
- 4: Scanner had not been connected.

5.2.10 unsigned short Scanner_SetSymbolIdentifierStatus(int index, BOOL bActive)

This function is used to enable or disable code block.

Parameters

index:[IN] which barcode to be change.

bActive: [IN] FALSE means to disable this barcode Symbol identifier, TRUE means to enable it.

Return Values

- 0: succeed.

3: Timeout

4: Scanner had not been connected.

6: Not supported symbol.

Remark

Refer Scanner_EnumSymbolIdentifierClass function or IscpRsSymbologies structrue.

5.2.11 unsigned short Scanner_GetSymbolIdentifierStatus(int index, BOOL* bActive)

This function is used to get specific barcode Symbol Identifier status.

Parameters

index:[IN] which barcode to be check. Must be equal or more than 0

bActive: [OUT] a output pointer to point out the barcode Symbol Identifier status.

Return Values

0: Succeed.

3: Timeout

4: Scanner had not been connected.

6: Not supported symbol.

5.2.12 unsigned short Scanner_EnumSymbolIdentiferClass(IscpRsSymbologies* pSymbologies, DWORD* count)

This function is used to acquire all symbol identifier class.

Parameters

pSymbologies: [OUT] A buffer pointer to IscpRsSymbologies structure array.

count: [IN/OUT] When in, it specifies the buffer size; when out, it specifies the real size of copied to buffer.

Return Values

It specifies the total number of symbol.

Remark

The following is the structure's declaration.

```
typedef struct {  
    int      index;  
    BOOL     bAction;
```

LPCSTR string;
BYTE type;
} IscpRsSymbologies;
index: Symbol's index.
bAction: This can specify whether the symbol is in action or not.
string: Symbol's string.
type: It can specify the symbol is 2D or 1D.

5.2.13 unsigned short Scanner_Trigger(void)

This function is used to send a trigger signal to Scanner.

Parameters

None

Return Values

- 0: Succeed.
- 3: Timeout
- 4: Scanner had not been connected.
- 6: Not supported triggering mode.

5.2.14 unsigned short Scanner_UnTrigger (void)

This function is used to send a signal to cancel trigger Scanner.

Parameters

None

Return Values

- 0: Succeed.
- 3: Timeout
- 4: Scanner had not been connected.
- 6: Not supported triggering mode.

Chapter 6 MSR SDK

6.1 Background

Provide a DLL to operation MSR device. DLL file name is DTMSrAPI.dll. LIB file name is DTMSrAPI.lib. Header file name is DTMSrAPI.h.

6.2 Function

6.2.1 unsigned short Msr_Connect(int port)

This function is used to establish host connection with the MSR.

Parameters

port: [IN] the number of serial port used to connect to MSR.

Return Values

0: Succeed.

1: Interface port not available

2: No device found

6.2.2 unsigned short Msr_Disconnect (void)

This function is used to close host connection with the MSR.

Parameters

None

Return Values

0: Succeed.

3: Timeout

4: MSR had not been connected.

6.2.3 unsigned short GetFirmwareInfo(char *name, long nameBufLen, char *version, long verBufLen)

This function is used to get the MSR firmware information.

Parameters

name: [OUT] pointer to a buffer that will be filled in with product name id.

nameBufLen: [IN] name buffer length.

version: [OUT] pointer to a buffer that will be filled in firmware version.

verBufLen: [IN] version buffer length.

Return Values

- 0: Succeed.
- 3: Timeout
- 4: MSR had not been connected
- 10: Invalid input buffer.
- 11: Input buffer length too small

6.2.4 unsigned short Msr_GetPortParameters(int *baud, int * databits, int *parity, int* stopbits)

This function is used to get current serial port communication parameters.

Parameters

- baud:[OUT] pointer to Baudrate, such as 9600,19200.
- databits: [OUT] pointer to Databits, such as 7,8
- parity: [OUT] pointer to Parity, such as 0,1
- stopbits: [OUT] pointer to Stopbits, such as 1,2.

Return Values

- 0: Succeed.
- 3: Timeout.
- 4: MSR had not been connected.

Remark

The parity predefined:

Index	0	1	2	3	4
Parity	NOPARIT	ODDPARITY	EVENPARITY	MARKPARITY	SPACEPARITY

6.2.5 unsigned short Msr_SetPortParameters(int baud, int databits, int parity, int stopbits)

This function is used to set serial port communication parameters to MSR. This will update MSR's firmware parameters.

Parameters

- baud: [IN] baud rate, such as 9600,19200.
- databits: [IN] databits parameter, such as 7,8

parity: [IN] parity parameter, such as 0,1

stopbits: [IN] stopbits parameter, such as 1,2.

Return Values

0: Succeed.

3: Timeout.

4: MSR had not been connected.

Remark

Index	0	1	2	3	4
Parity	NOPARIT	ODDPARITY	EVENPARITY	MARKPARITY	SPACEPARITY

6.2.6 unsigned short Msr_SetDataReceiverCallback (MsrDataCallback revFun)

This function is used to identify the host application a data receive callback function, when data received from MSR, that function will be called.

Parameters

revFun:[IN] a function pointer.

Return Values

0: Succeed.

1: Function pointer is invalid.

Remark

The callback function is defined as following.

```
typedef void (__stdcall*MsrDataCallback)(TRACKDATA *pTrackData,  
                                         DWORD dwTrackCount)
```

Parameter pTrackData is a structure pointer. This structure is defined as following.

```
typedef struct tagTRACKDATA  
{  
    char startSentinel;  
    char endSentinel;  
    LPSTR lpszDataBuf;  
    DWORD dwDataLen;
```

```
}TRACKDATA;
```

startSeninel: Starting sentinel character of the track.

endSeniel: Ending sentinel character of the track.

lpszDataBuf: Track data's buffer.

dwDataLen: Track data's length.

Paramter dwTrackCount specifies the total number of TRACKDATA structure.

6.2.7 unsigned short Msr_ParseEnable (BOOL bParse, struct PAESEDATAFORMAT *parseFormat, int parseCount)

This function is used to parse MSR data by given parameters.

Parameters

bParse: [IN] to identify enable or disable parse function. FALSE for disable, TRUE for enable.

parseFormat:[IN] a pointer to parsing data format structure.

parseCount:[IN] the count of parsing data format structure.

Return Values

0: Succeed.

1: Parsing data format invalid.

Remark

The parsing data structure is defined as following.

```
typedef struct tagPARSEDATAFORMAT
{
    char startSentinel;
    char endSentinel;
}PARSEDATAFORMAT;
```

Chapter 7 FRID SDK

7.1 Background

Provide a DLL to operation RFID reader. At present, we have two kinds of RFID reader, MStar and Digi_Link. So this DLL will support the two readers. DLL file name is DTRfidAPI.dll. LIB file name is DTRfidAPI.lib. Header file name is DTRfidAPI.h.

This Dll is not thread safe guaranteed.

7.2 Function

7.2.1 int RFID_Connect(int port)

This function is used to connect RFID reader.

Parameters

port: [IN] Specify the serial port connected to RFID reader.

Return Values

0: failed.

1: Successful.

7.2.2 int RFID_Disconnect(void)

This function is used to disconnect the connection with RFID reader.

Parameters

None

Return Values

1: The connection is successfully disconnected.

1. 0: Failed to disconnect. This may be due to that the connection is not exist, or RFID reader is busy. RFID_SetUidReceiverCallback

7.2.3 int RFID_SetUidReceiverCallback(RFIDUidCallback pUidFun)

This function is used to identify the host application RFID's UID callback function. When RFID is detected, the function will be called.

Parameters

pUidFun: [IN] A function pointer.

Return Values

0: failed.

1: Successful.

Remark

The callback function is defined as following.

```
typedef void (*RFIDUidCallback) (const char* pszUidArray,  
                                unsigned int nCount)
```

Parameters:

pszUidArray: [IN] A zero ended ASCII string. This string contains an array of uid and type pair. For example: "4EFA178A:ISO1443A, 33FA178B:ISO1443A "

":" is used between Uid and its type. "," is used between different Uid:Type pairs.

nCount:[IN] The total number of RFIDs found.

7.2.4 int RFID_ReadBlockData (const char* pszUid, const char* pszPassword, unsigned int nBlkIdx, unsigned char* pBuf, unsigned int* pBufSize)

This function is used to read one block's data.

Parameters

pszUid: [IN] RFID's UID, it identifies which RFID will be read. Only zero ended ASCII string is valid.

pszPassword: [IN] Block's password. Only zero ended ASCII string is valid.

nBlkIdx:[IN] Block's index.

pBuf: [OUT] Pointer to a buffer for receives data returned from RFID.

pBufSize:[IN/OUT] When in, this variable contains the buffer size of parameter lpBuffer. When out, this variable contains returned data size.

Return Values

0: failed.

1: Successful.

7.2.5 int RFID_GetFirmwareInfo(char* pVerBuf, int nBufSize)

This function is used to get RFID reader's firmware information.

Parameters

pVerBuf: [OUT] A pointer to firmware info.

nBufSize: [IN] The size of pVerBuf.

Return Values

0: failed.

1: Successful.

Chapter 8 RTC SDK

8.1 Background

Use CeRunAppAtTime() API to set alarm time. After suspend OS, when current time is the same as the alarm time, OS will be resumed.

8.2 Function

8.2.1 **int CeRunAppAtTime(TCHAR* pwszAppName, SYSTEMTIME* lpTime)**

Please reference MSDN.