CTFPND-5
User Manual

Features :

- ARM926EJ CPU
- 7” 800x480 TFT
- Host USB x 1
- Device USB x 1
- SD/MMC card port
- 2-wire RS232 x 2
- 8-wire RS232 x 1
- GPIO x 8
- GPS
- GSM
- Audio
- WinCE support

History of Version

Version	Contents	Date	Note
01	New Version	2010/03/27	SPEC.
01a	Pp91. GetGPOutput ->SetGPOutput	2010/6/3	

Table of content

1. General Information

This chapter provides basic information of CTFPND-5 module and consists of the following:

1.1. Introduction

CTFPND-5 is a our Mobile Data Terminal (MDT) product. Equipped with a GPS/GSM module, the system can easily transmit the location info to GSM/control center for dispatching services when in need.

CTFPND-5 builds upon a Win CE operation system. With WinCE's versatile and solid application support, designer may custom-designed application program with a short development timeframe. In terms of I/O of CTFPND-5, there are 8 customizable buttons, 3 RS-232, 1 host USB, 1 device USB, 4 sets of general-purpose I/O (GPIO).

Here is the comprehensive product line of CTFPND-5 family :

Order Information

Part No.	Touch Panel	Battery	LCD	Touch Panel
CTFPND-5-0			-30℃~80℃	
CTFPND-5-1	*		-30℃~80℃	-30℃~80℃
CTFPND-5-2		*	-30℃~80℃	
CTFPND-5	*	*	-30℃~80℃	-30℃~80℃
CTFPND-5-4			-20℃~70℃	
CTFPND-5-5	*		-20℃~70℃	-20℃~60℃
CTFPND-5-6		*	20℃~70℃	
CTFPND-5-7	*	*	20℃~70℃	-20℃~60℃

1.1.1. Packing contents

Check your package for the following items:

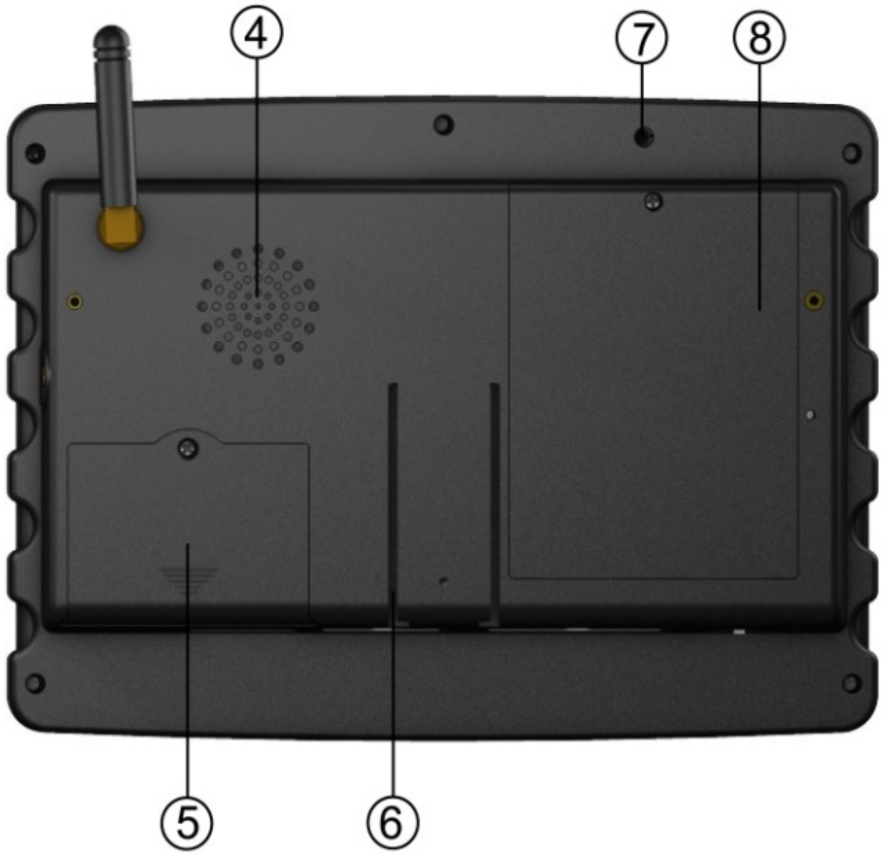
- CTFPND-5 module
- Holder & mount
- Mini-USB to USB cable
- GSM antenna
- Power and I/O cables (only for sample stage)
- Serial cable (only for sample stage)
- Rechargeable battery (options)

1.1.2. System outline

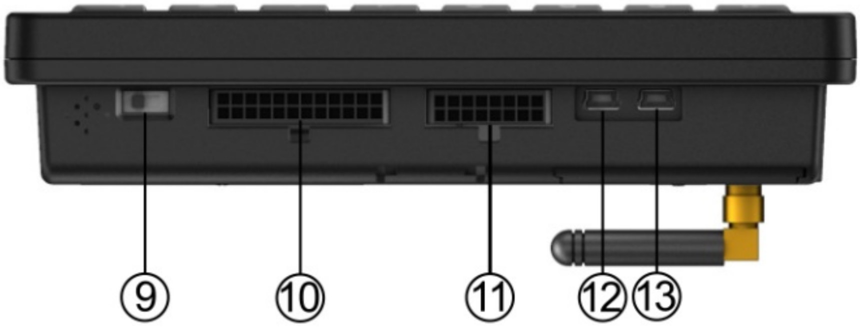
#	Name	Description
1	LED Indicator	Indicates CTFPND-5 Status
2	Microphone	Supports GSM for voice communication
3	Push Buttons	Supports 8 customized menu buttons (none system default)

#1 Status of LED Indicator

	LED color	Description
	Red	Power on
	Orange	Low power
	Green	GPS transmitting signal
	Yellow	GSM receiving signal

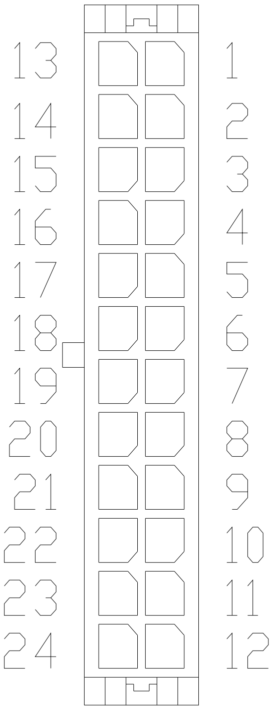


#	Name	Description
4	Speaker	For audio replay
5	SIM Card port	GSM/SIM Card slot
6	Mount & holder	Fix the module in a car
7	GPS antenna	Supports external GPS Antenna(One build-in GPS supported)
8	Battery box	Installs optional lithium battery



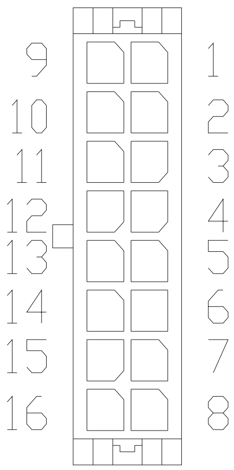
#	Name	Description
9	Power switch	Turn on/off power
10	Power & I/O ports	Power input，4x photo-coupler input，4x photo-coupler output
11	Serial I/O	2-wire RS232 x 2, 8-wire RS232 x 1
12	Host –USB	External USB1.0 host for file access
13	Device-USB	Connect to PC for data sync by using *ActiveSync*

#10 : Pin assignment for power and I/O ports

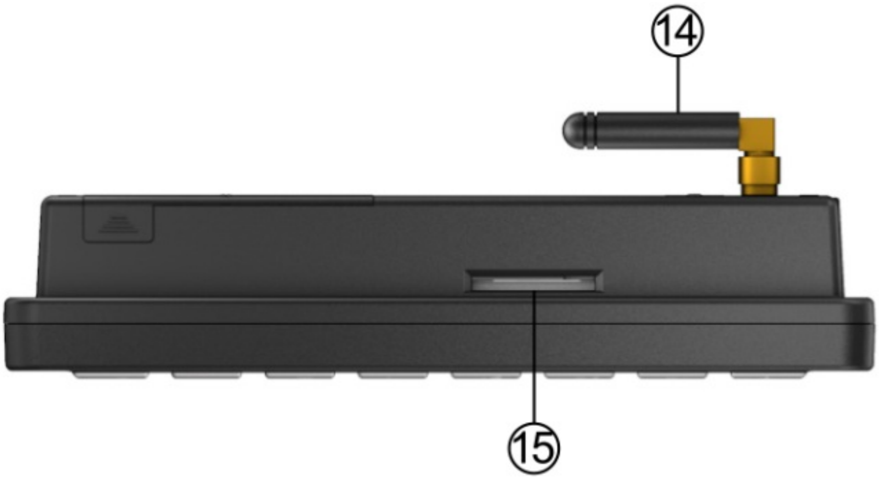


13	DCIN	1	DCIN
14	GND	2	GND
15	NC	3	VIG
16	VIO	4	VIO
17	IN1	5	OUT1
18	IN1_GND	6	OUT1_ND
19	IN2	7	OUT2
20	IN2_GND	8	OUT2_GND
21	IN3	9	OUT3
22	IN3_GND	10	OUT3_GND
23	IN4	11	OUT4
24	IN4_GND	12	OUT4_GND

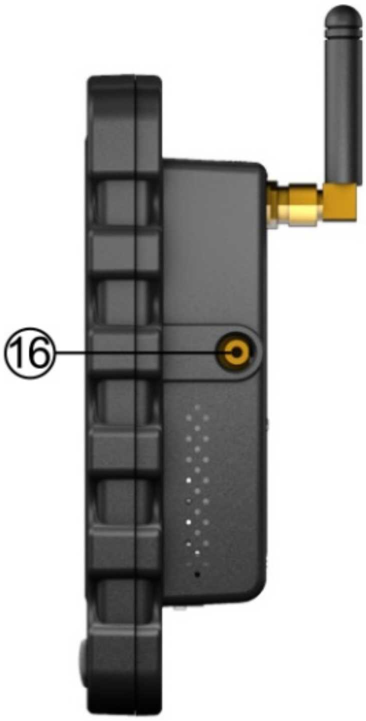
#11 : Pin assignment for serial I/O ports



9	TXD5T	1	GND
10	RXD5T	2	TXD1T
11	RTS5T	3	GND
12	CTS5T	4	RXD1T
13	DTR5T	5	NC
14	DCD5T	6	TXD3T
15	RI5T	7	NC
16	DSR5T	8	RXD3T



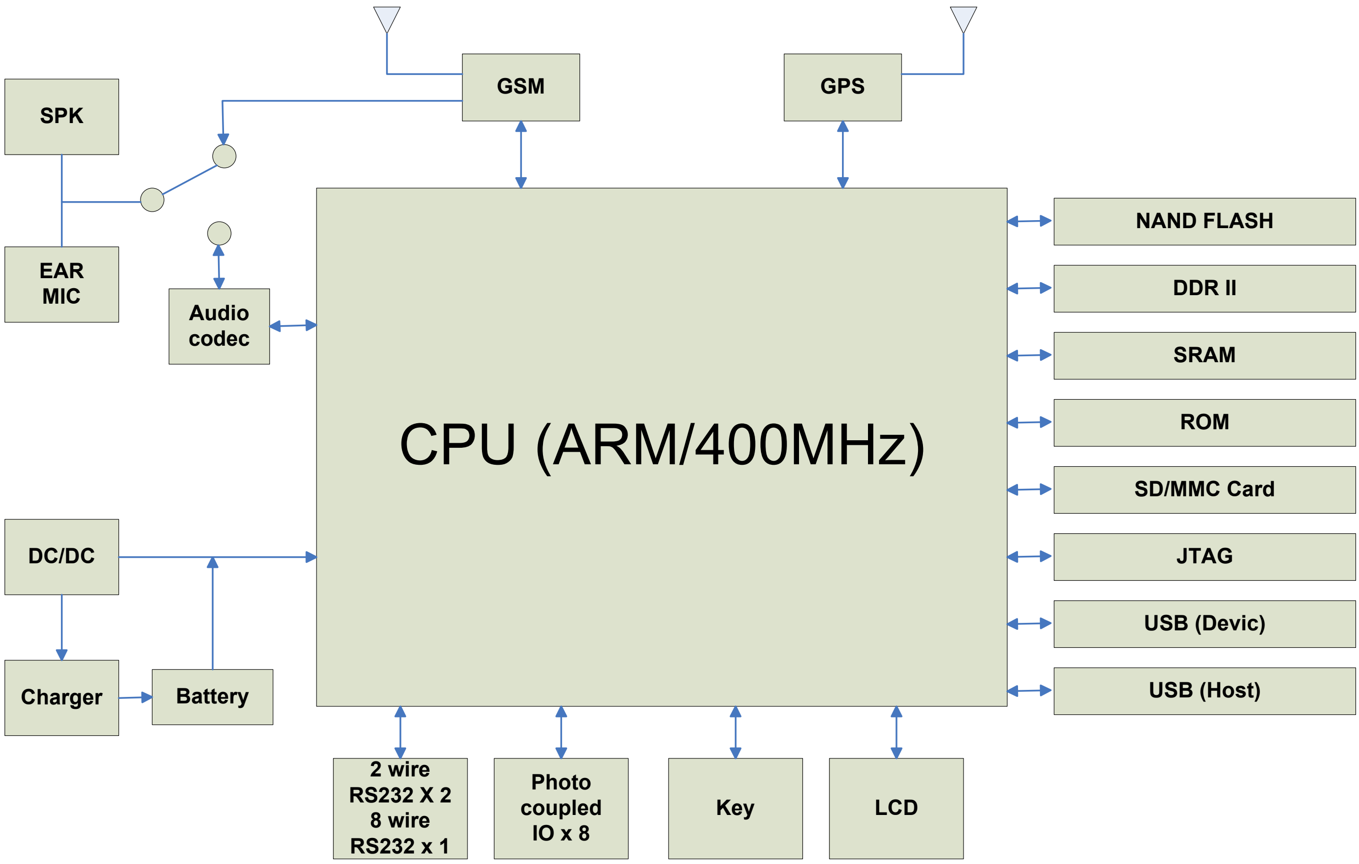
#	Name	Description
14	GSM Antenna	Connects to GSM
15	SD Card port	SD/MMC Card slot (up to 4GB max)



#	Name	Description
16	Earphone slot	Connects to stereo earphone

1.2. Specification

1.2.1. System functional blocks



1.2.2. System Specifications

Parameter	Specifications
CPU	• Samsung S3C2416X 400MHz • 32 bit RISC architecture ARM926EJ CPU core
System Memory	• 16-bit 64MB/133MHz DDR2 memory • 512K Byte SRAM
Storage Device	2GB NAND Flash
Series Port	• 2-wire RS232 x 2 • 8-wire RS232 x1
USB	• 1x USB device (USB2.0) • 1XUSB host (USB1.0)
GPIO	Supply 4 photo-coupler input and 4 photo-coupler output
Audio	• Support DSP based processing stereo codec with SNR 102 dB DAC/differential microphone input • Dual channels 2 watts speaker output
Integrated Modules	• GPS Module (LEA-5S) • GSM Module (SIEMENS MC-55)
Supply OS	WinCE 5.0(default)
LCD Size	7” TFT LCD
LCD Response	800 x 480 RGB
LCD Brightness	400 cd/m²
Power Supply	DC9V~DC28V
Operating Temperature	-30℃ ~ +80℃

1.2.3. GPS Module Specifications

Parameter	Specifications			
Receiver Type		50 Channels		
		GPS L1 frequency, C/A code		
		GALILEO Open Service L1 frequency		
Time-To-First-Fix	Sky View	Open Sky[2]		Indoor[3]
	Module	ALL	LEA-5H/LEA-5S	LEA-5A
	Cold Start(Autonomous)	29 s		
	Hot Start(Autonomous)	<1 s	10 s	TBD
	Aided Start	<1 s	10 s	10 s
	Reacquisition	<1 s	10 s	TBD
Sensitivity	Tracking & Navigation	-160 dBm		
	Acquisition & Reacquisition	-160 dBm		
	Cold Start(Autonomous)	-145 dBm		
Horizontal Position Accuracy[4]	Autonomous	<2.5 m		
	SBAS	<2.0 m		
Accuracy of Time pulse Signal	RMS	50 ns		
Max Navigation Update Rate		4 Hz		
Dynamics		≤ 4 g		
Operational Limits	Velocity	515 m/s (1000 knots)		

[2] All satellites at -130dB
[3] All satellites at -155dBm
[4] 50%, 24 hours static, -130dBm

1.2.4. GPS Antenna Specifications

Parameter	Specifications
Patch Specifications	
Center Frequency	1575.42±3 MHz
Bandwidth	6 MHz
Polarization	Linear
S11	<-15 dB
Max Gain	-0.5(typ.)(144,162) dBi
Frequency Temperature Coefficient	0±20 ppm/℃
Filter/LNA Specifications	
Gain	19±3 Db (DC=3.0V)
Noise Figure	1.5 dB(typ) (DC=3.0V)
Output V.S.W.R	2.0 max (DC=3.0V)
Current(DC=3.0V±0.01V)	3.5 ±1.5 mA
Overall Specifications	
Center Frequency	1575.42±1.023 MHz(When covered with a radome on LAN ground plane.)
Gain at Zenith	18 dBi typ (for ground 32x8.7 mm)
Output Impedance	50 ohm
Output VSWR	2.0 typ.
Operation Voltage	3.0 ±03 V

1.2.5. GSM Module specifications

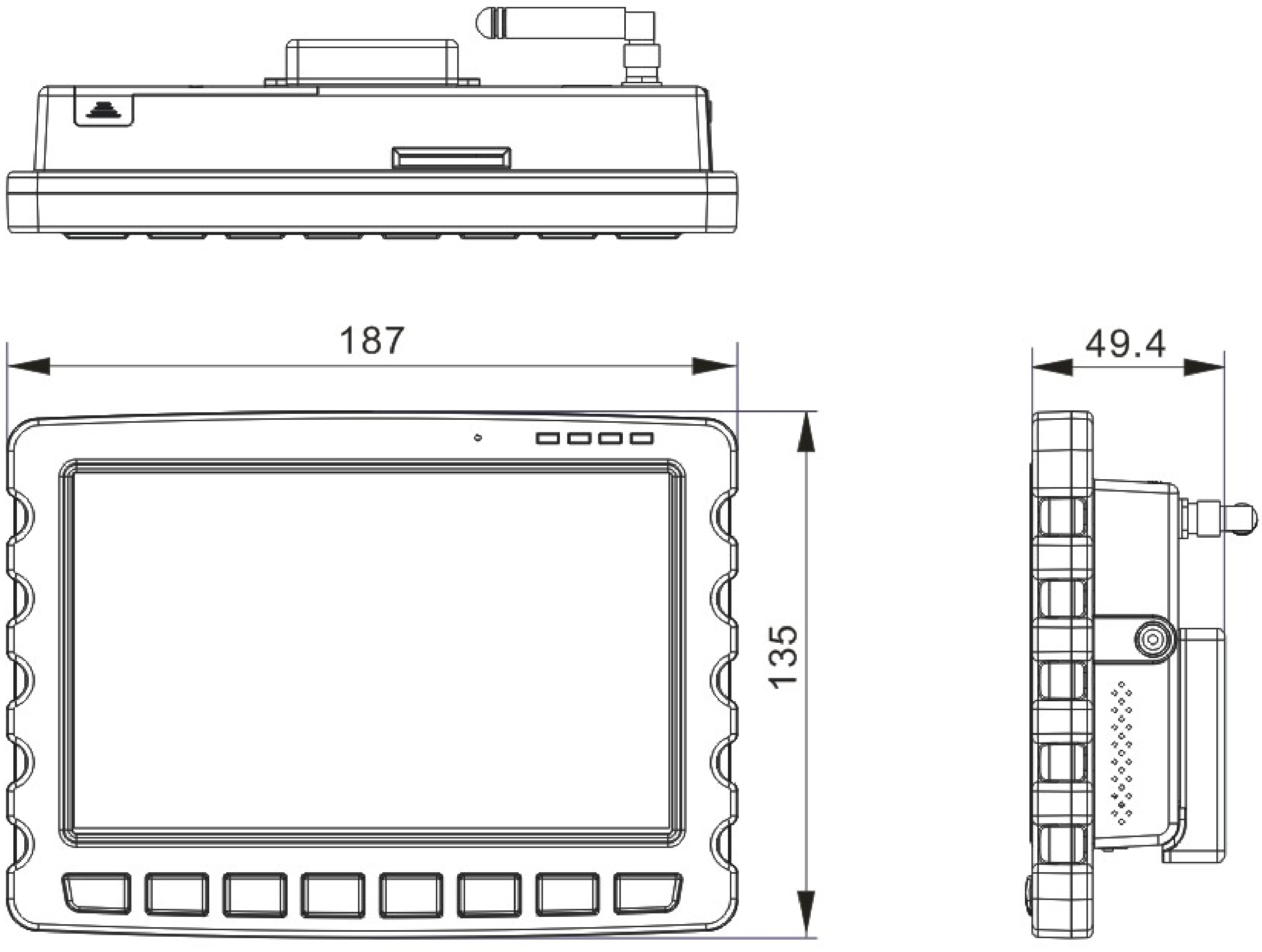
Parameter	Specifications
Frequency band	• MC55 Tri-band: EGSM 900, GSM 1800, GSM 1900 • Compliant to GSM Phase 2/2+
GSM class	Small MS
Transmit power	• Class 4 (2W) at EGSM 900 and GSM 850 • Class 1 (1W) at GSM 1800 and GSM 1900
GPRS connectivity	• GPRS multi-slot class 10 • GPRS mobile station class B
Ambient operating temperature according to IEC 60068-2	• Normal operation: -20℃ to +55℃ • Restricted operation: -25℃ to -20℃ and +55℃ to +70℃ • Automatic thermal shutdown: ≦ -25℃ and ≧+70℃ When an emergency call is in progress automatic temperature shutdown is deferred
Humidity	Max. 90% relative humidity
GPRS data	• GPRS data downlink transfer: max. 85.6 kbps • GPRS data uplink transfer: max. 42.8 kbps • Coding scheme: CS1, CS2, CS3 and CS4 • MC55/MC56 Support the two protocols PAP (Password Authentication Protocol) and CHAP (Challenge Handshake Authentication Protocol) commonly used for PPP connections. • Support of Packet Switched Broadcast Control Channel (PBCCH) allows you to benefit from enhanced GPRS performance when offered by the network operators.
CSD data	• CSD transmission rates: 2.4, 4.8, 9.6, 14.4 kbps, non-transparent, V.110 • Unstructured Supplementary Services Data (USSD) support
SMS	• MT, MO, CB, Text and PDU mode • SMS storage: SIM card plus 25 SMS locations in the mobile equipment • Transmission of SMS alternatively over CSD or GPRS. Preferred mode can be user-defined
TCP/IP stack	Internet services: TCP, UDP, HTTP, FTP, SMTP, POP3 Access by AT commands
FAX	Group 3: Class 1, Class 2
SIM interface	• Supported SIM card: 3V • External SIM card reader has to be connected via interface connector
External antenna	Connected via 50 Ohm antenna connector or antenna pad
Audio interfaces	Two analog audio interfaces, one digital audio interface(DAI)
Audio features	Speech codec modes: • Half Rate (ETS 06.20) • Full Rate(ETS 06.10) • Enhanced Full Rate(ETS 06.50/06.60/06.80) • Adaptive Multi Rate(AMR) Hands free operation • Echo cancellation • Noise reduction

1.2.6. GSM antenna specifications

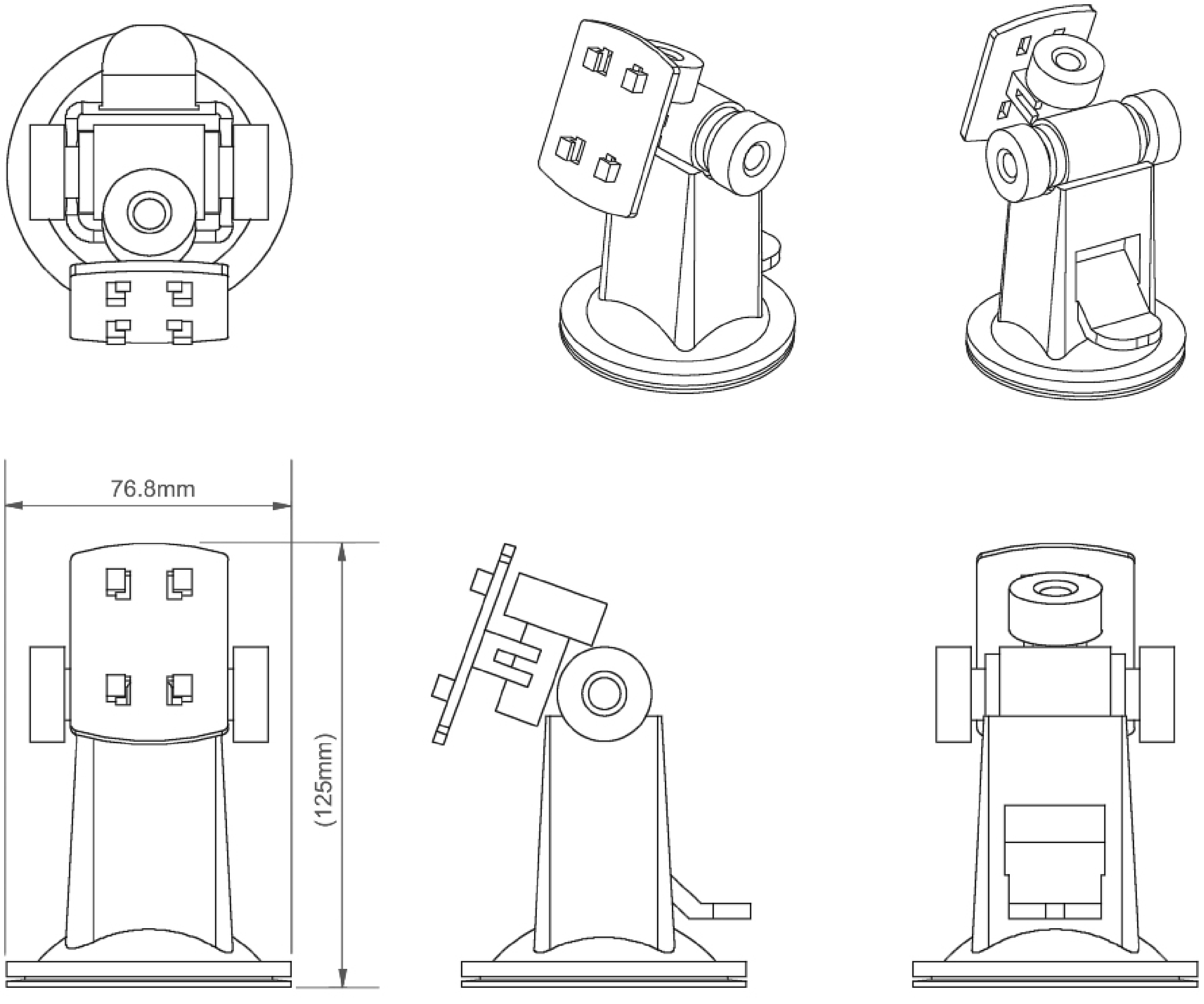
Parameter	Specifications
Frequency Range	880~960 MHz and 1710~1990 MHz
Impedance	50 ohm
VSWR	$\leqq$3.5
Gain	0 dBi (Max)
Polarization	Vertical
Radiation pattern	Near omni-directional in the horizontal plane

1.3. Mechanical specifications

1.3.1. Mechanical specifications of module



1.3.2. Mechanical specifications of Holder & Mount

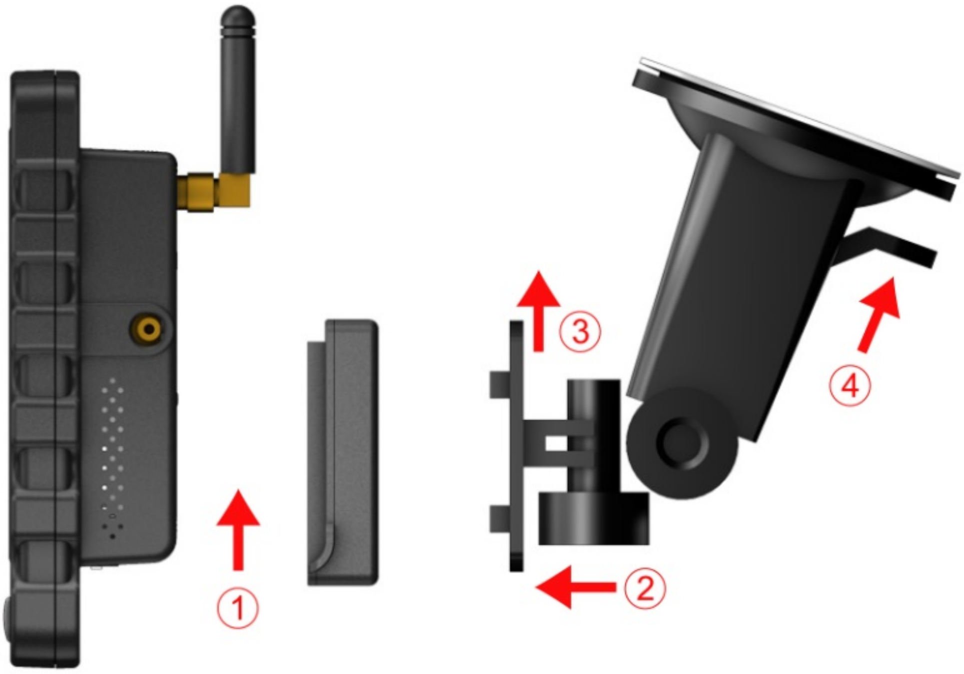


2. CTFPND-5 Installation guide

The chapter provides guidance for hardware installation.

2.1. Install a sucker & mount

Follow the sequence shown below, install CTFPND-5 on the sucker & mount and adhere the set onto the front-window. Carefully choose an appropriate adhering spot such that driving safety is ensured.



2.2. Install a GSM antenna

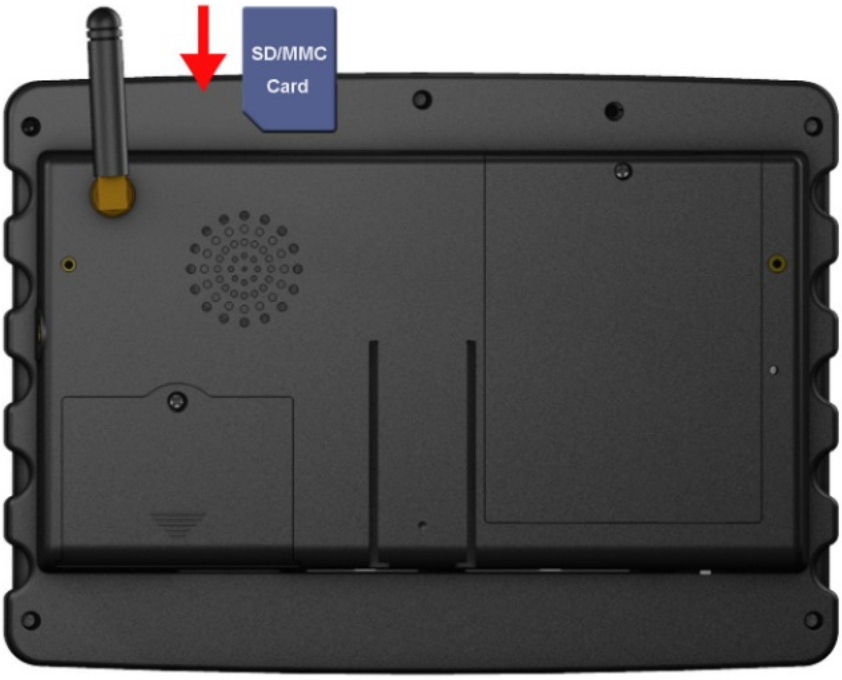
Please screw up the antenna into the position as highlighted.

2.3. Install a SIM card

Open the SIM Card cover as shown below, push left the SIM card holder, lever up and place in the SIM card into its fixed position. Place the cover into its original position.

2.4. Install a SD card

As shown in the following photo, plug in the SD card. To remove a SD card, first make sure the SD is not in the middle of reading process, then push on the SD card to release it.



2.5. Install external GPS antenna

CTFPND-5 comes with a build-in GPS antenna. In case better reception is required, users may purchase GPS antenna with MMCX connector as appropriate and connect it the module as indicated. Stretch the antenna outside the car or outdoors with clear line of sight.

2.6. Install battery

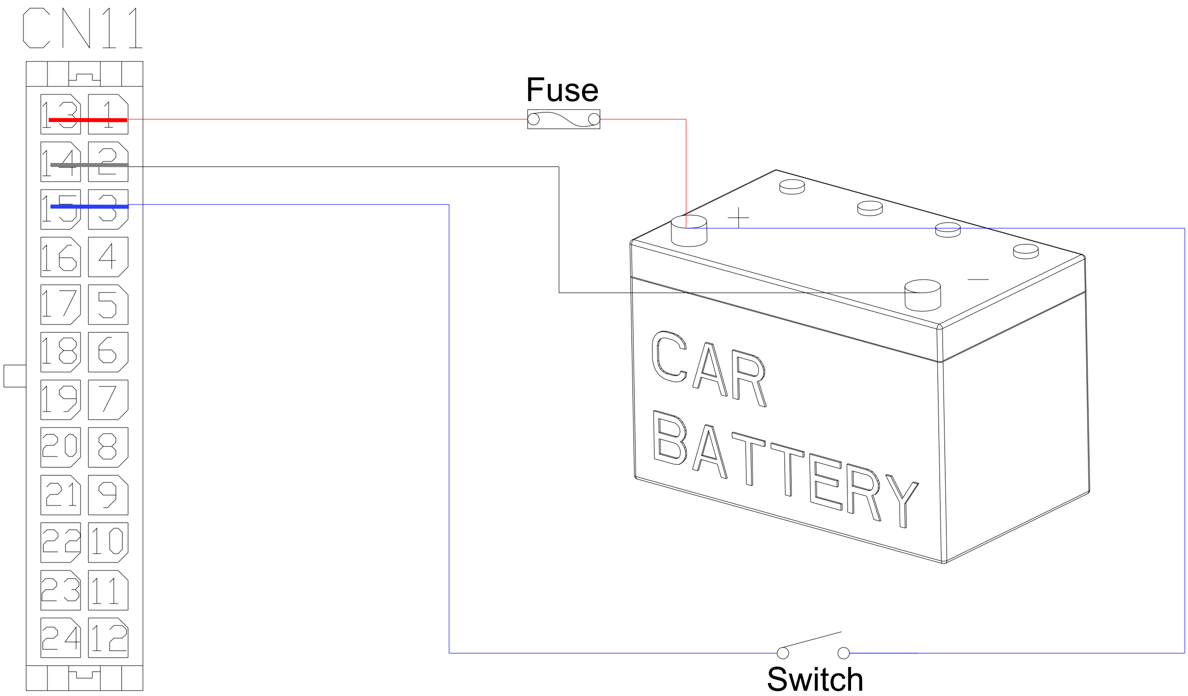
Turn off CTFPND-5 before installing battery. First, unscrew the cover of battery box and remove the cover. Then install the battery connector into the socket as indicated, and place the battery as appropriate. Place the cover back and screw it up to complete the battery installation.

Note that only a 3.7V/2700mA Lithium battery is applicable, and typically with 300 to 350 recharge cycles.

2.7. Connect Power

The operating voltage of CTFPND-5 ranges from 9 to 28 volt and no more than 1.5A current. Short pin 1-13, 2-14,3-15 as illustrated. Connect pin 1 and 13 to the positive contact of 12/24v car battery, and pin 2 and 14 to the negative. As colored blue , connect Pin 3 and Pin15 to a car switch (a single-throw switch as indicated) and the other end of switch to positive end of car battery.

After power connection is done, switch onCTFPND-5 to boot up.



Note:

A quick troubleshooting hint – If CTFPND-5 does not boot up as expected, please check if the fuse (with black housing) on the power and I/O ports are broken. If yes, replace them with 5*20mm / 1.5A/ 250V fuses. Otherwise, return the CTFPND-5 module for repair.

2.8. CTFPND-5 power status

When no (optional) battery is installed in CTFPND-5, the power is support by car battery – that is, car switch needs to be "on" position and power switch of CTFPND-5 need to be on to boot up. If optional battery is applicable, simply power on CTFPND-5 module to boot up using the battery power. To charge the battery, turn the car switch on. The charging will be automatically stopped as soon as the batter is fully charged.

Status Table					
Mode	Car switch	CTFPND-5 Power switch	CTFPND-5 Battery Status	CTFPND-5 RTC Battery Status	CTFPND-5 Host status
1	OFF	OFF	No charging	Charging	Off
2	ON	OFF	Charging	Charging	Off
3	ON	ON	Charging	Charging	On
4	OFF	ON	No charging	Charging	On

RTC (Real Time Clock) battery:

Mode 1- When car switch and power switch of CTFPND-5 are both OFF, the RTC battery inside CTFPND-5 still supply power to CPU-RTC and GPS-memory. During this mode, RTC battery still get slightly charged even for long time no use.

RTC battery can sustain GPS-memory for up to one month. If the next power on occurs later than one month, it may require more time (up to 30 minutes) for GPS re-positioning to re-initialize the GPS-memory.

Note that if GPS positioning always takes more than 30 minutes for every boot up, it may due to the failure of RTC battery. It is recommended to replace the RTC battery.

Mode 2- Turn car switch on while keep CTFPND-5 off, CTFPND-5 battery and RTC battery will get charged from car battery.

Mode 3- Turn car switch on, then turn CTFPND-5 is on. Same as mode 2, CTFPND-5 battery and RTC battery will get charged from car battery.

Mode 4- Turn car switch off, Same as mode 1, CTFPND-5 battery will NOT get charged, and RTC battery will get charged by car battery.

Just a reminder, the RTC battery always get charged by car battery for all switch combinations.

3. How to test on CTFPND-5

This chapter provides a step guide to test I/O on CTFPND-5 module and it is broken down into:

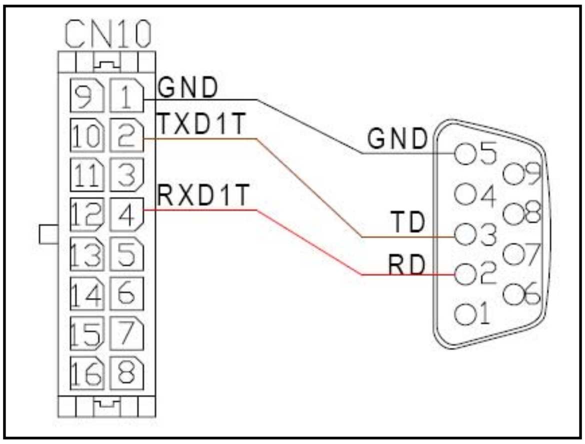
3.1 Serial port test
3.2 GSM test
3.3 GPS test
3.4 GPIO test
3.5 Keypad test

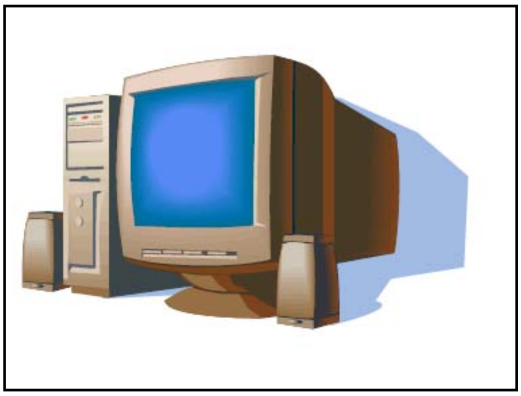
3.1. Serial Port test

3.1.1. Connect Serial Port to PC

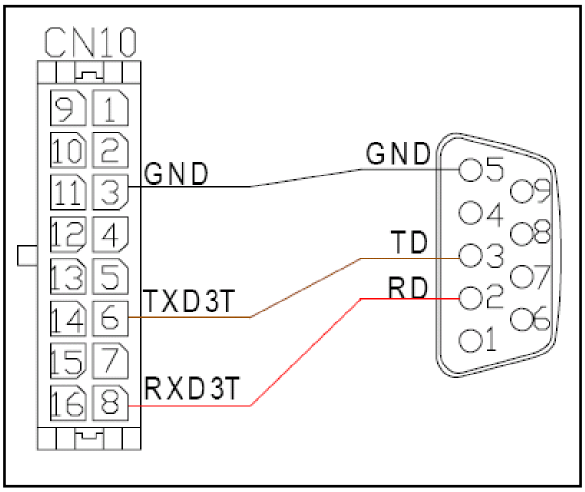
CTFPND-5 supports two sets of 2-wire RS232 and one set of 8-wire RS232 for serial interface to a PC.

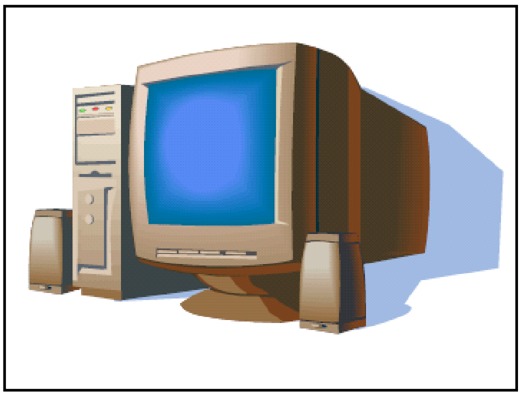
1st set of 2-wire RS232 (Debug Port) - Via a RS-232 serial cable, connect the 1st set to COM port of PC. This port is defaulted to a debug port and not for external use. Its signal level is at +/- 12 volt.



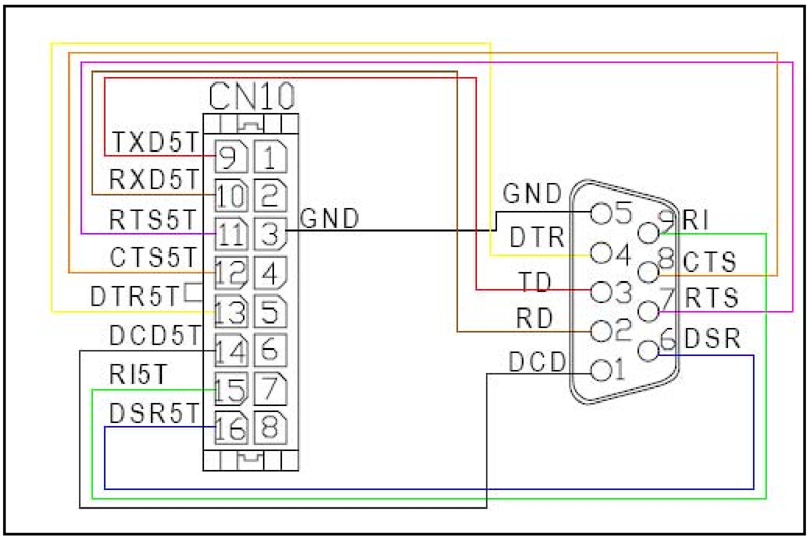


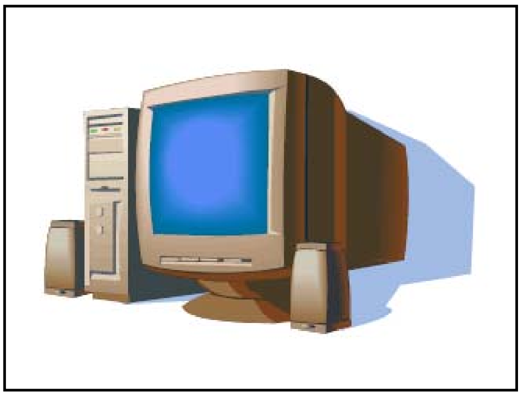
2nd set of 2-wire RS232 (COM3) - Via a RS-232 serial cable, connect the 2nd set to COM port of PC. This port is defaulted to COM3 with a signal level of +/- 12 volt.





8-wire RS232 (COM6) – Via a serial cable, connect the 8-wire RS232 to COM port of PC. This port is defaulted to COM3 with a signal level of +/- 12 volt.





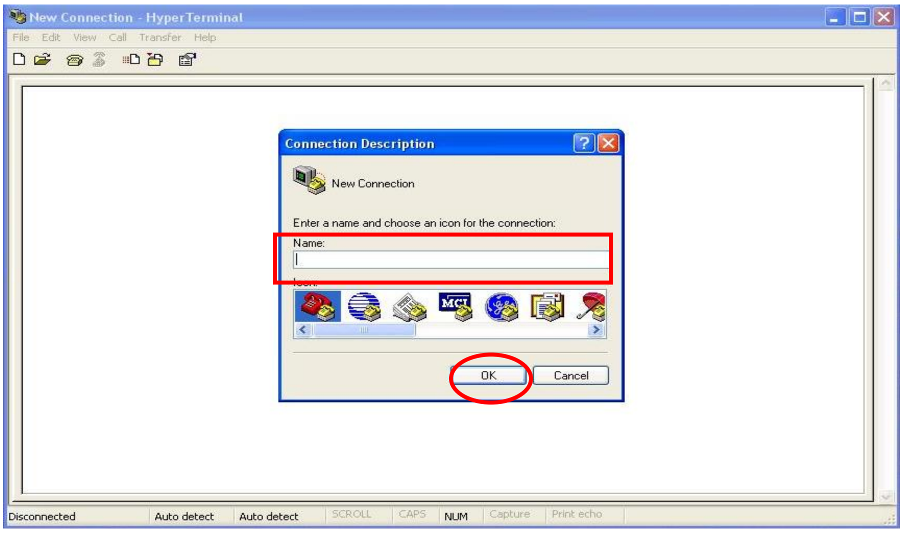
The following table shows the pin assignment mapping for all available serial ports.

Table of RS232 Pin Assignment							
RS232 DB9 Connector		1st set of 2-wire RS-232 (Debug Port)		2nd set of 2-wire RS-232 (COM3)		8-wire RS-232 (COM6)	
Pin No	Pin Name	Pin No	Pin Name	Pin No	Pin Name	Pin No	Pin Name
1	DCD	—	—	—	—	14	DCD5T
2	RD	2	RXD1T	6	RXD3T	9	RXD5T
3	TD	4	TXD1T	8	TXD3T	10	TXD5T
4	DTR	—	—	—	—	13	DTR5T
5	GND	1	GND	3	GND	3	GND
6	DSR	—	—	—	—	16	DSR5T
7	RTS	—	—	—	—	12	RTS5T
8	CTS	—	—	—	—	11	CTS5T
9	RI	—	—	—	—	15	RI5T

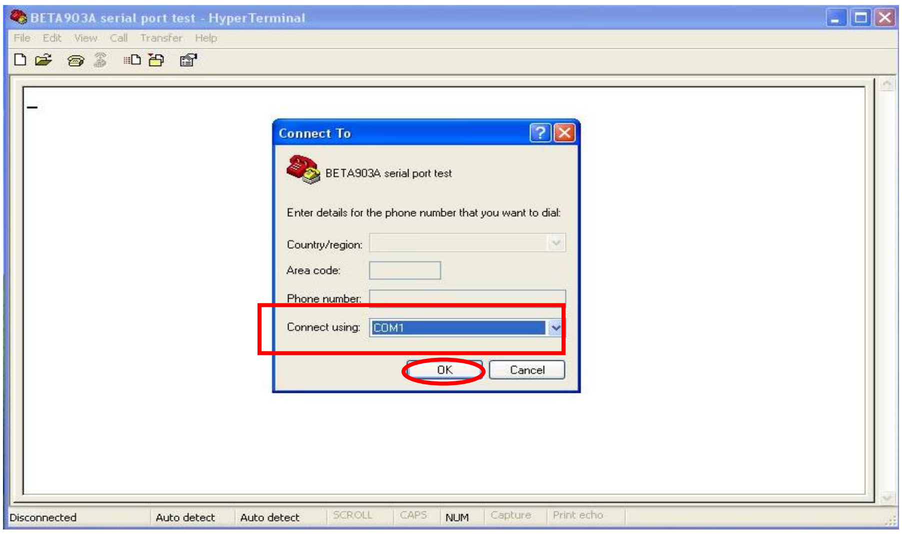
3.1.2. Serial port test

Upon completion of serial connection of CTFPND-5 to PC, users may use hyper-terminal (or other terminal emulator) to diagnose the communication link. Here is the step guide to install hyper-terminal at PC end.

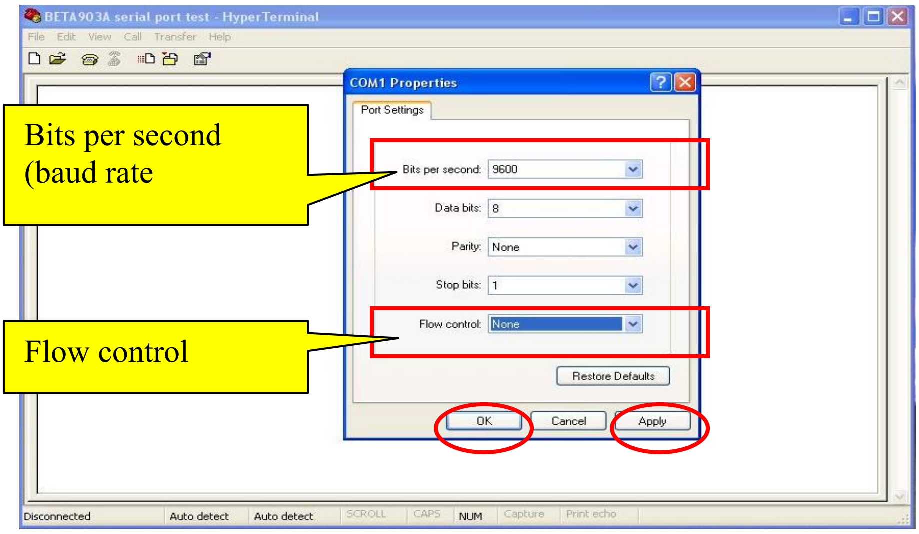
STEP1: Run the hyper-terminal on PC, then enter CTFPND-5, click on OK.



STEP2: A “Connect to” window pops up, on “Connect using”, select the applicable COM port, click OK. To find applicable COM port, Click Control panel – system – hardware - device manager -COM&LPT ports.

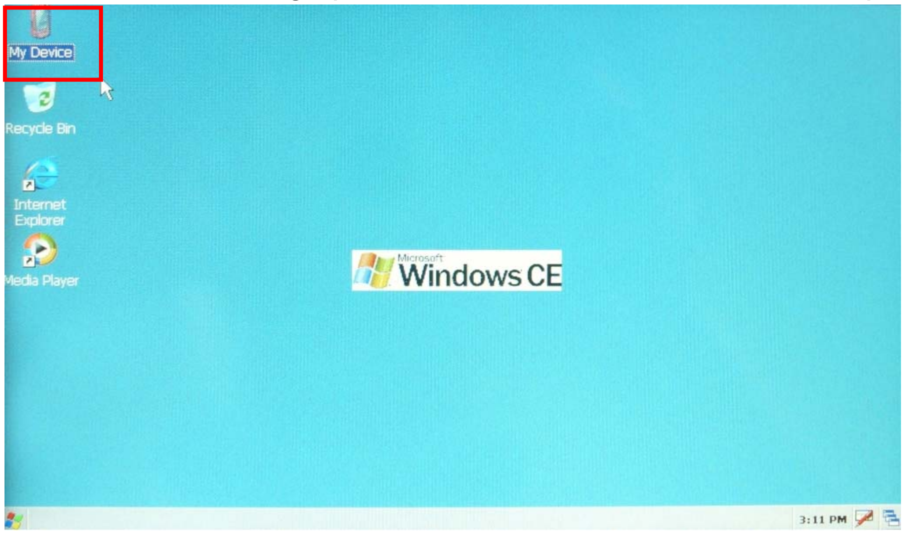


STEP3: A "COMx Properties" window pops up, make sure the "bits per second" (or baud rate) and "flow control" settings match with those on CTFPND-5. A typical baud rate of CTFPND-5 ranges from 9600 to 115200 and use "None" for flow control. Click on Apply, then OK.

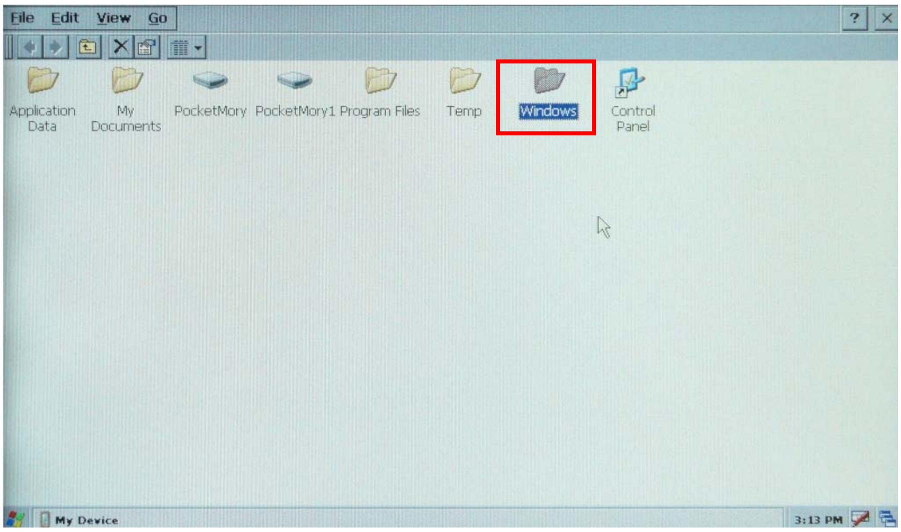


CTFPND-5 set up:

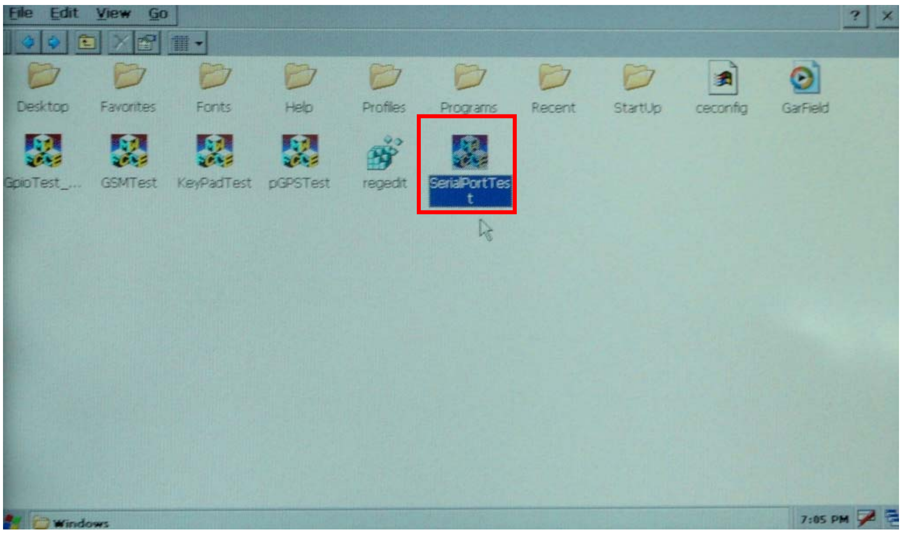
STEP1: After booting up CTFPND-5, on Windows CE desktop, click on ”My Device” ◦



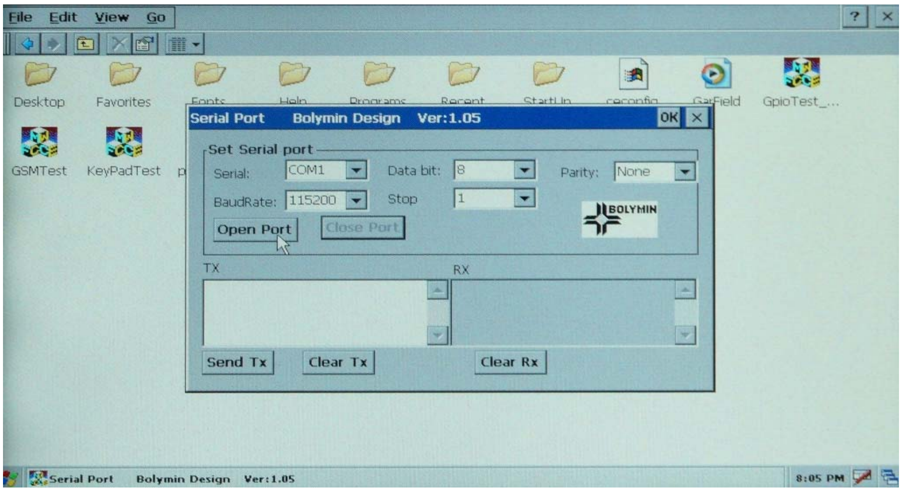
STEP2: Select “Windows” folder



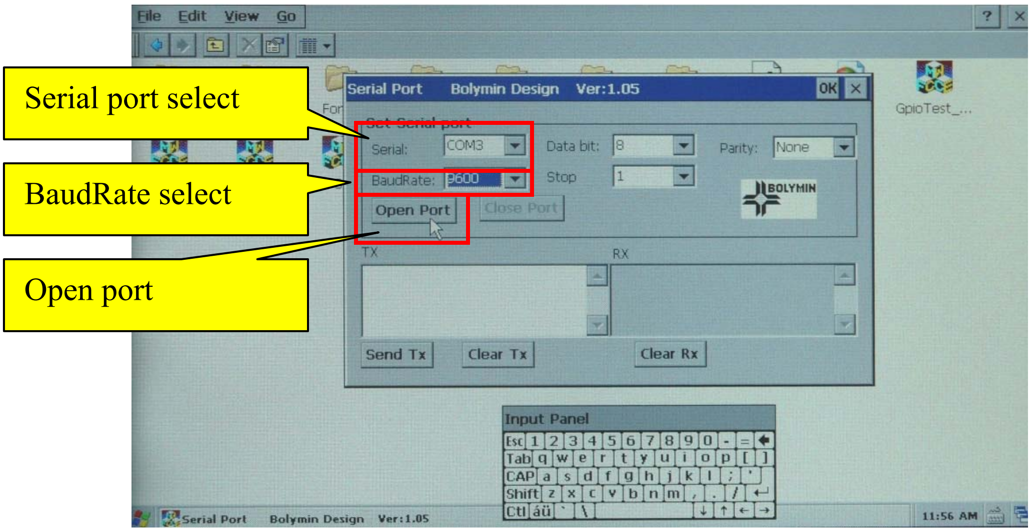
STEP3: Run SerialPortTest” program



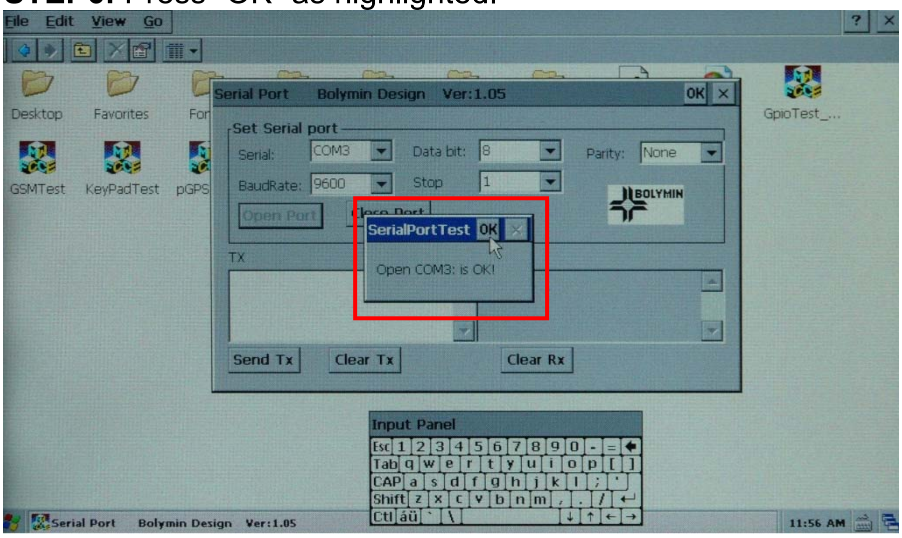
STEP4: A "SerialPortTest" running screen.



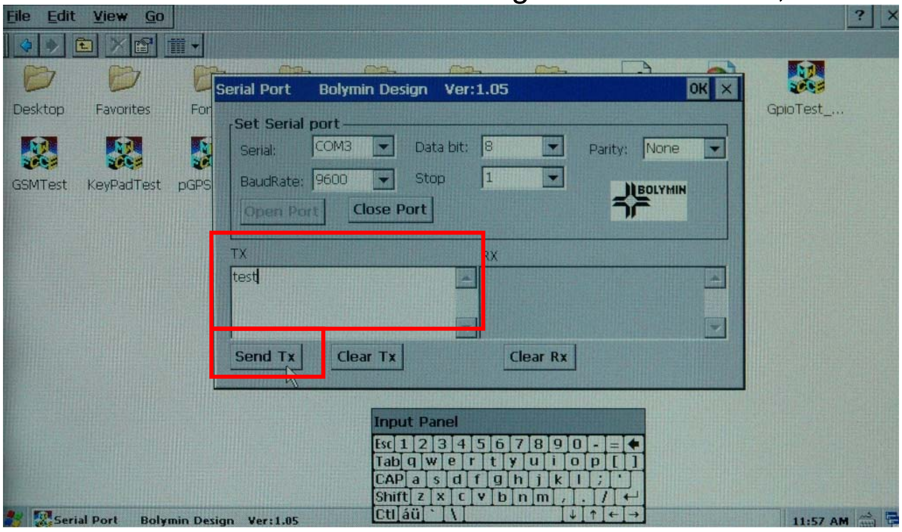
STEP5: Please make sure the baud rate setting on PC and CTFPND-5 are identical with a typical range of 9600 to 115200.then clock on "Open Port" to connect to PC.



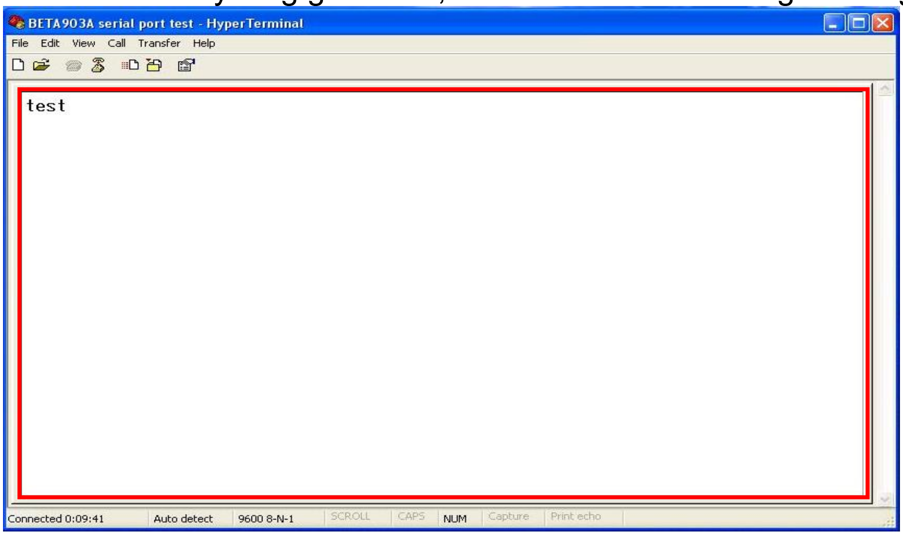
STEP6: Press "OK" as highlighted.



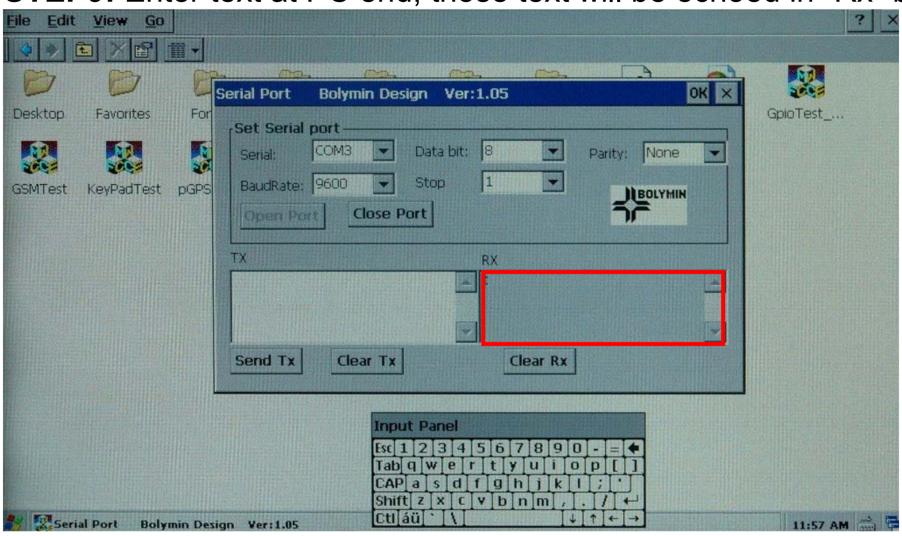
STEP7: Enter some trial text message on the "TX" box, then click on "Send Tx" .



STEP8: If everything goes fine, user will see a receiving message as the one transmitted.



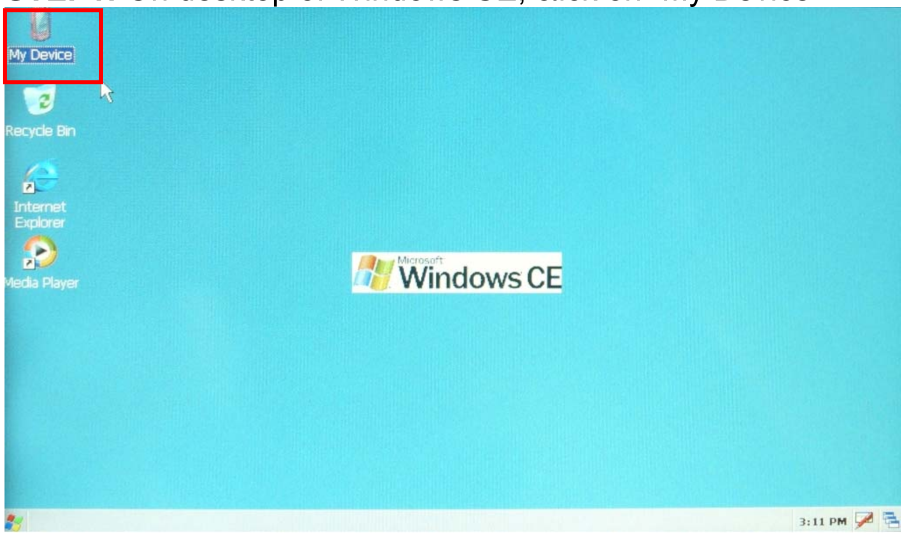
STEP9: Enter text at PC end, those text will be echoed in "Rx" box at CTFPND-5.



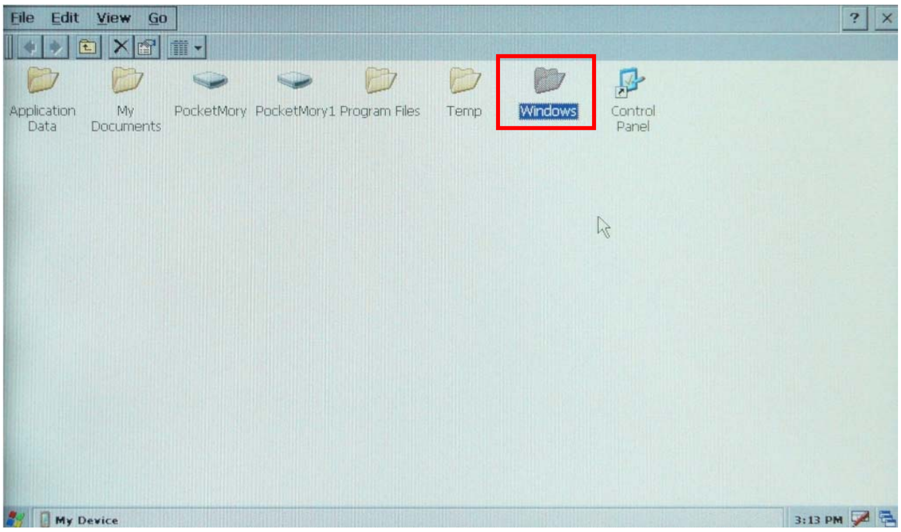
3.2. GSM test

Test procedure:

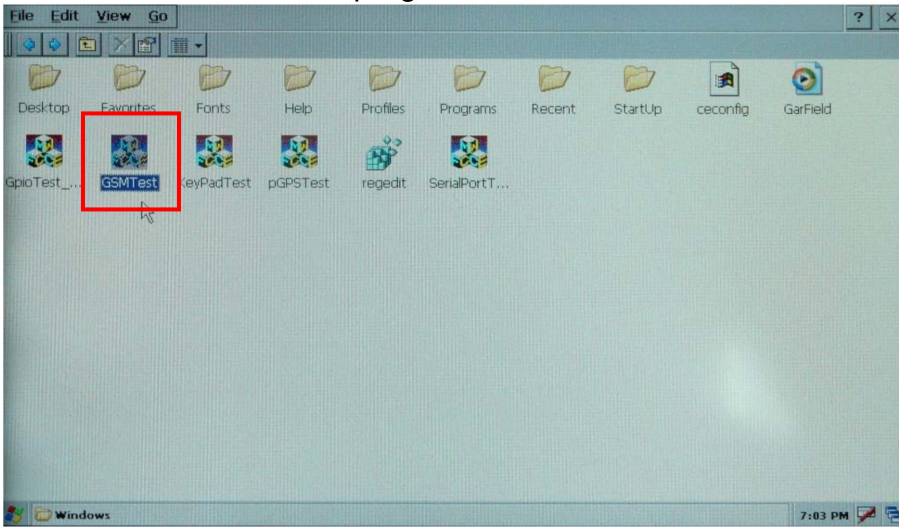
STEP1: On desktop of Windows CE, click on ”My Device”



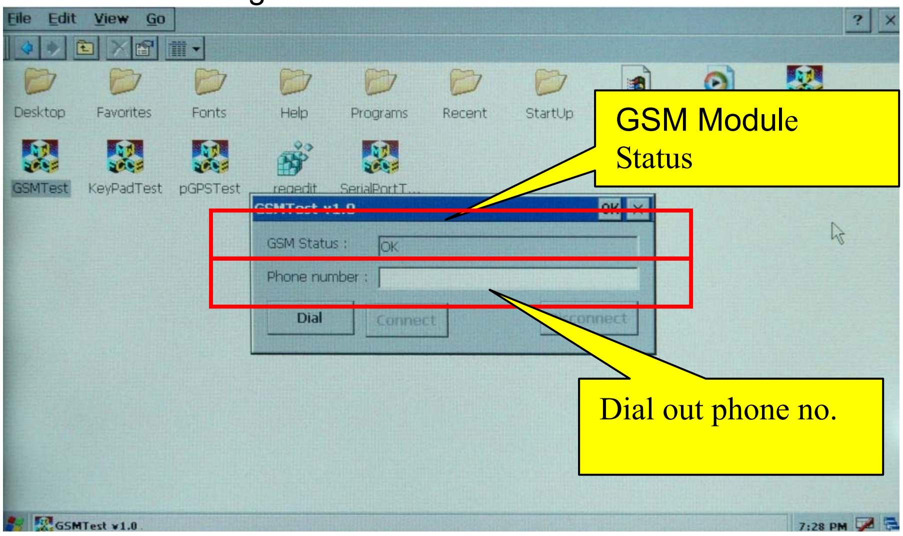
STEP2: Select ”Windows” folder



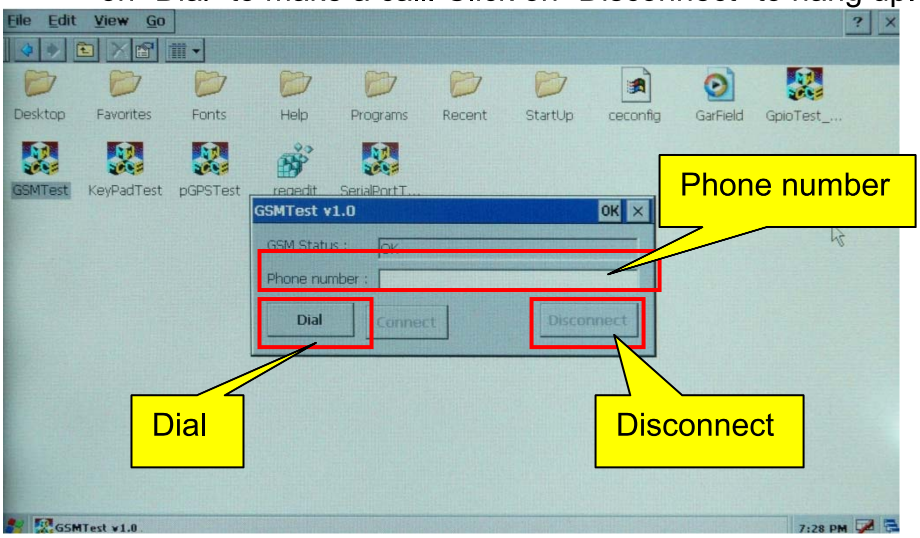
STEP3: Run "GSMTest" program.



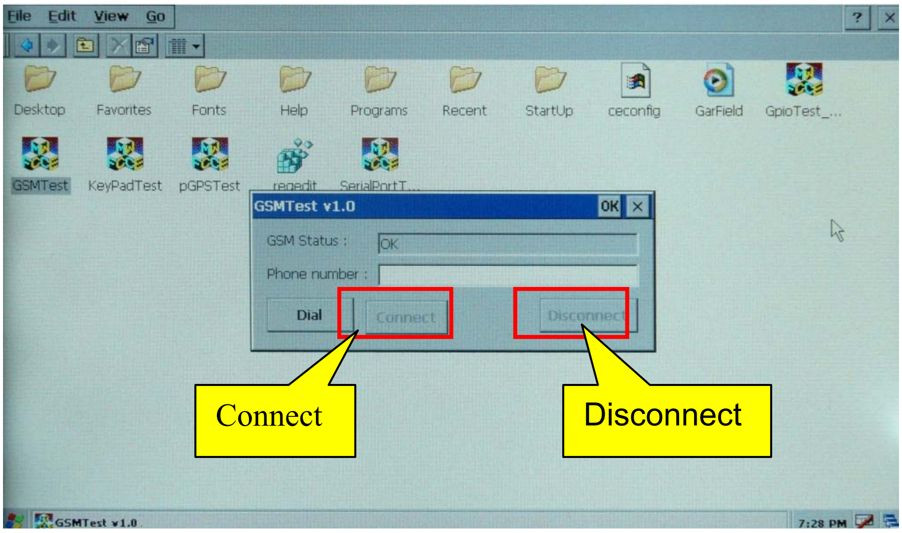
STEP4: Running screen of "GSMTest"



STEP5: Before dialing out, enter the phone number in the "Phone number" box, tehn click on "Dial" to make a call. Click on "Disconnect" to hang up.



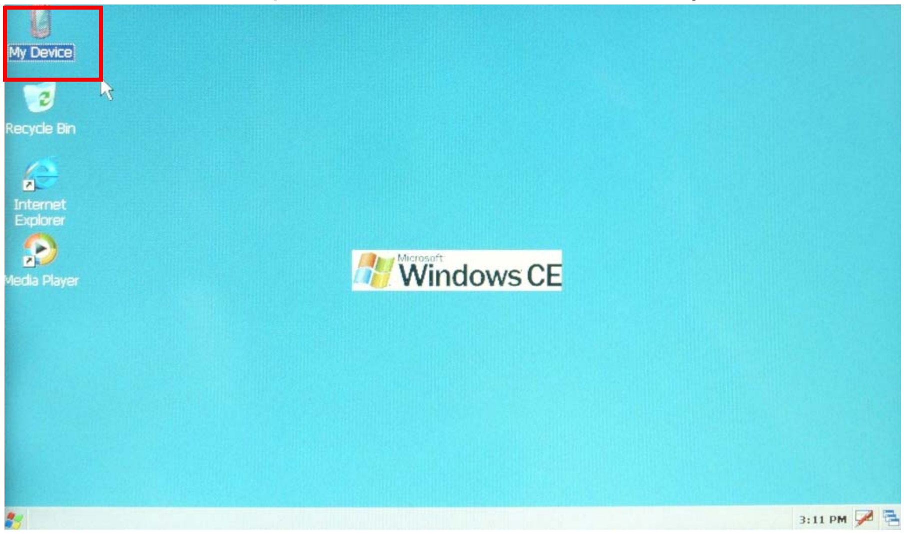
STEP6: CTFPND-5 will pop up a screen show a "RING" at status bar when an incoming call occurs. Click on "Connect" to pick up the phone. Click on "Disconnect" to hang up.



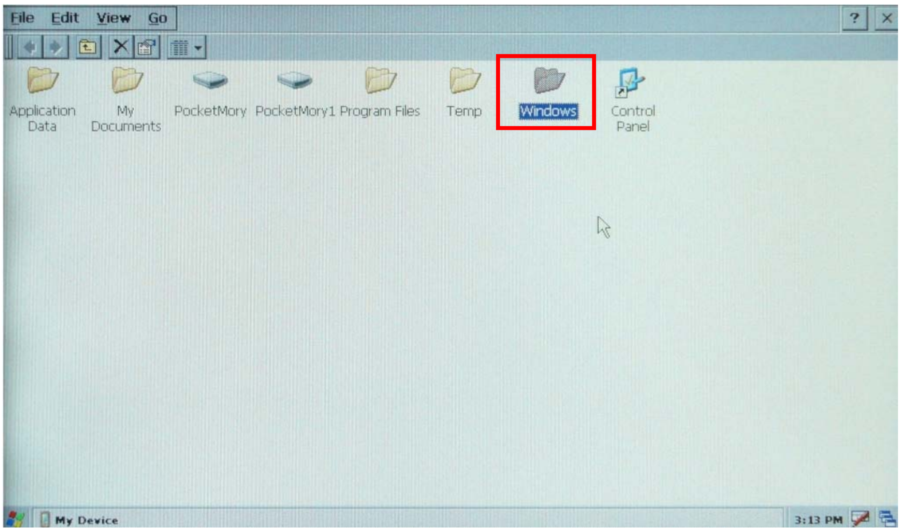
3.3. GPS test

Test Procedure:

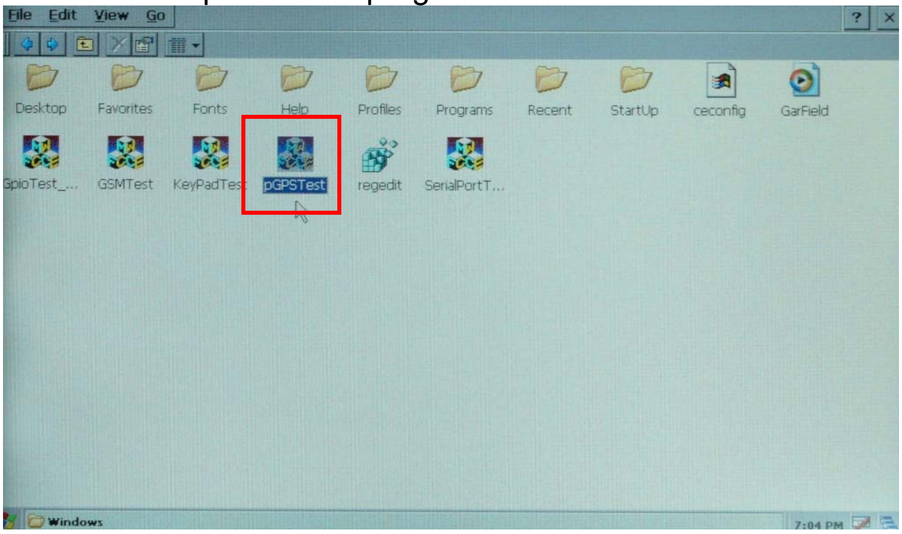
STEP1: On desktop of Windown CE, click on "My Device" 。



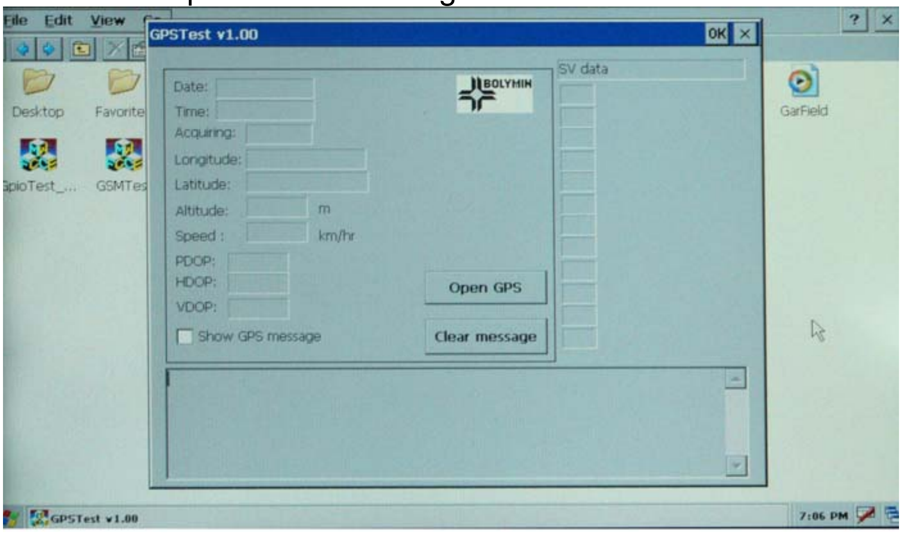
STEP2: Click on "Windows" folder



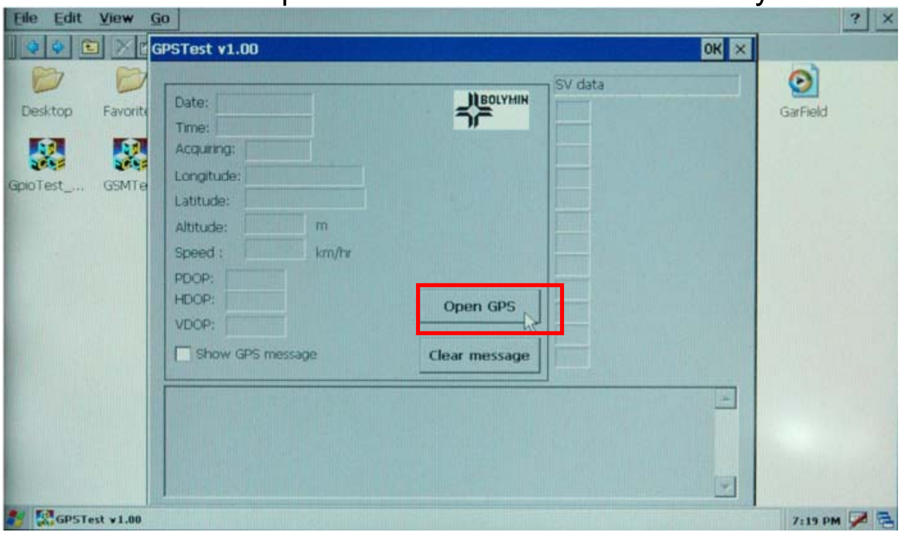
STEP3: Run "pGPSTest" program



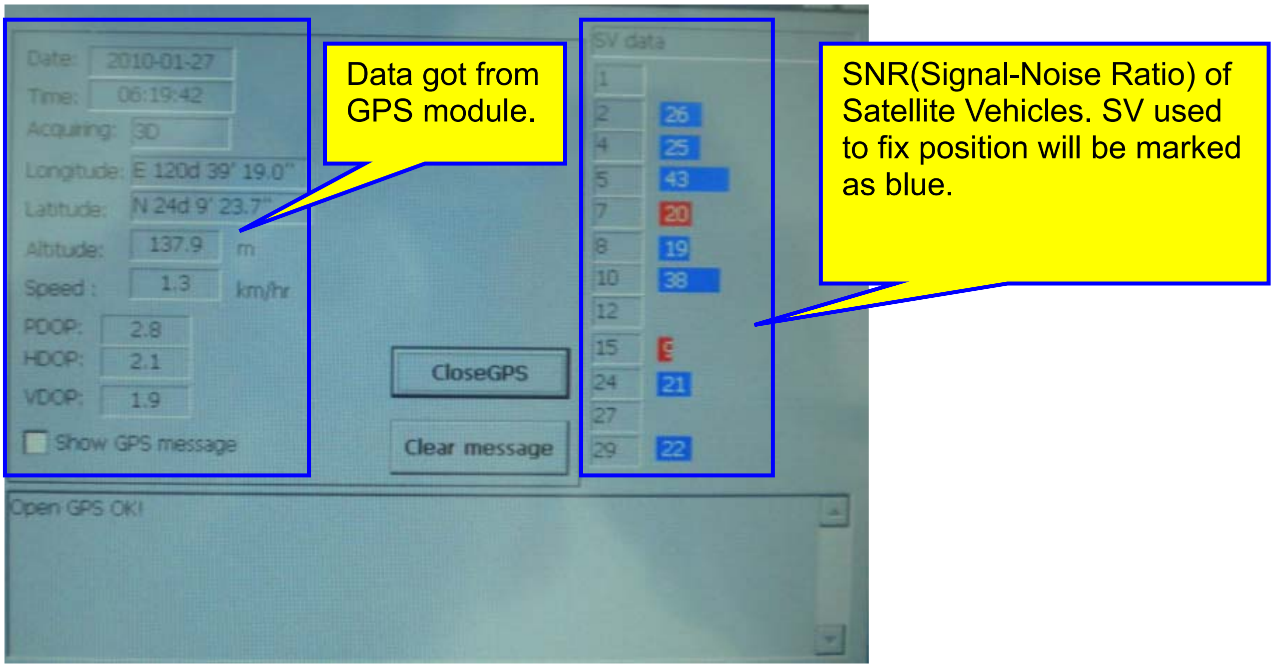
STEP4: A "pGPSTest" running screen



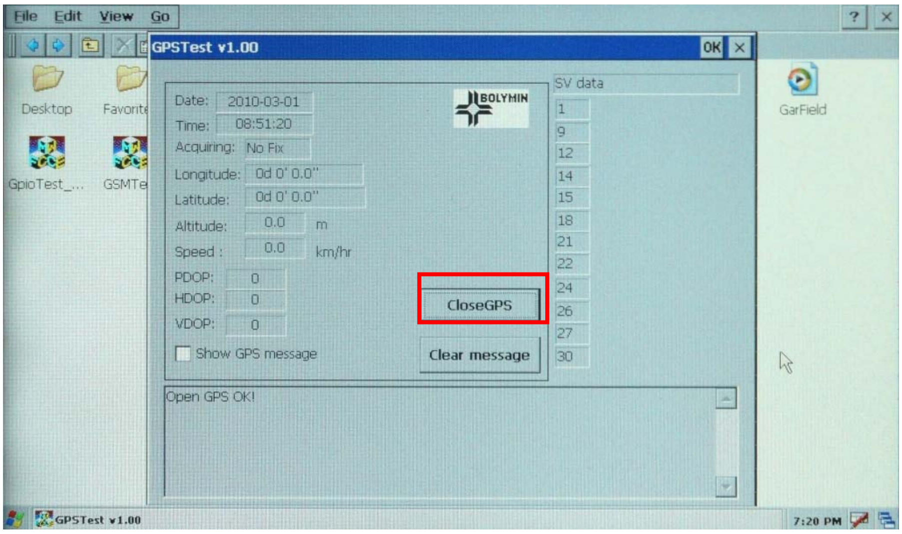
STEP5: Click on "Open GPS" to test GPS connectivity.



STEP6: If GPS satellite receiving is valid, the program will show as follows:



STEP7: Click on “Close GPS” to terminate GPS connection



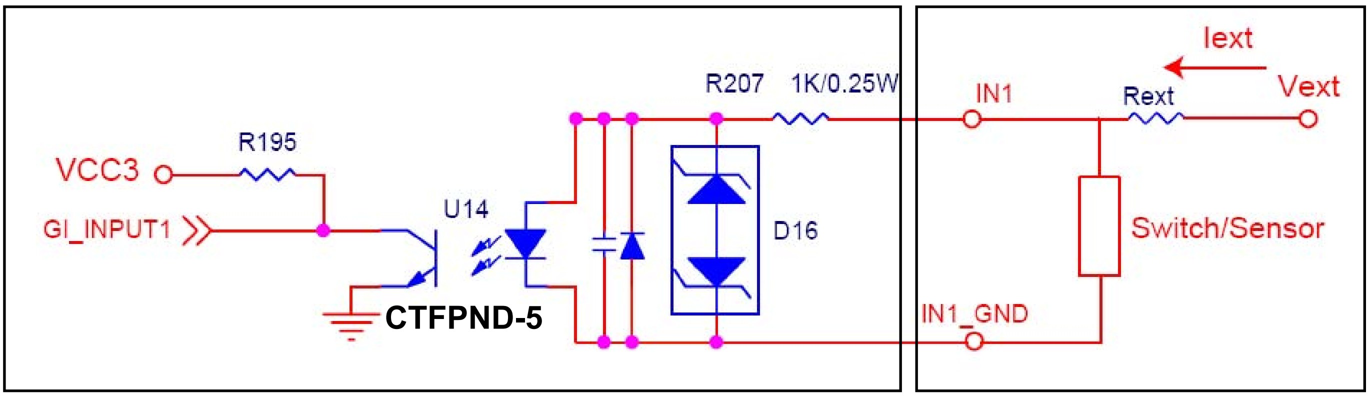
3.4. GPIO test

3.4.1. Recommended interfacing circuit to GPIO

CTFPND-5 supports 4 sets of photo-coupler inputs and 4 sets of photo-coupler outputs for general purpose inputs/outputs. Those I/Os are useful to read out signals and access control devices.

.

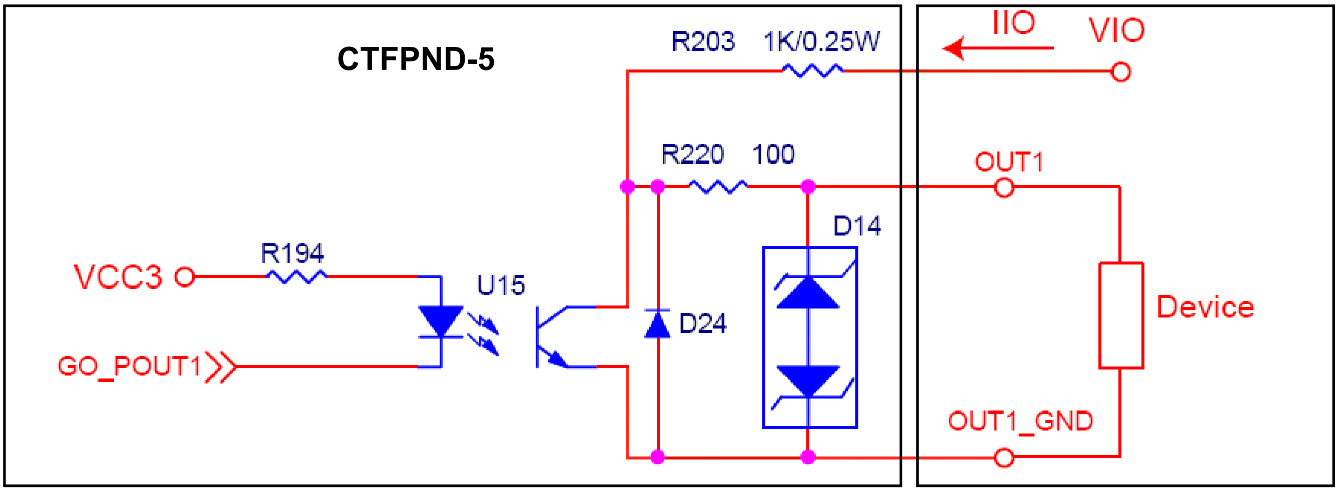
Input circuit examples:



Note: The value of Rext depends on that of Vext. The rule of thumb is keep Iext(min current) at a typical value of 5mA , and Iext(max) at 14mA. If Rext=0 , set Vext to 12 volt, then Iext will be 12mA。

Output circuit example: Shunt OUT1 and GND with a device as shown on the right side, and provide a power through VIO pin. Device will be turned on if GO_POUT1 is set to low. Among which GO_POUT1 is a software-defined parameter, please refer to Chapter 3.4 for details. On the other hand, Device will be turned off if GO_POUT1 is set to high.

. ◦



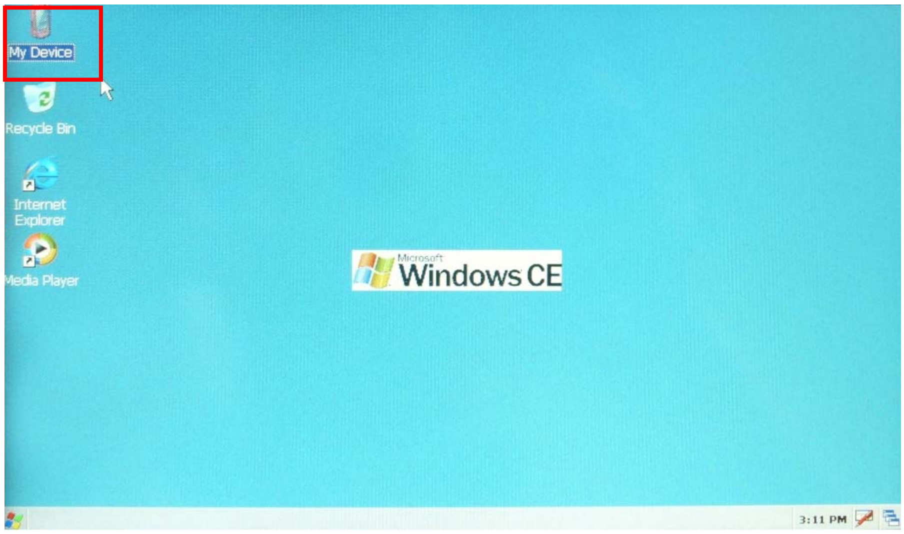
Note: The voltage and current value of OUT1 pin depends upon VIO and Rex, if any. Use caution to keep IIO(min current) to 5mA and IIO(max) to 14mA as rated.

3.4.2. GPIO test

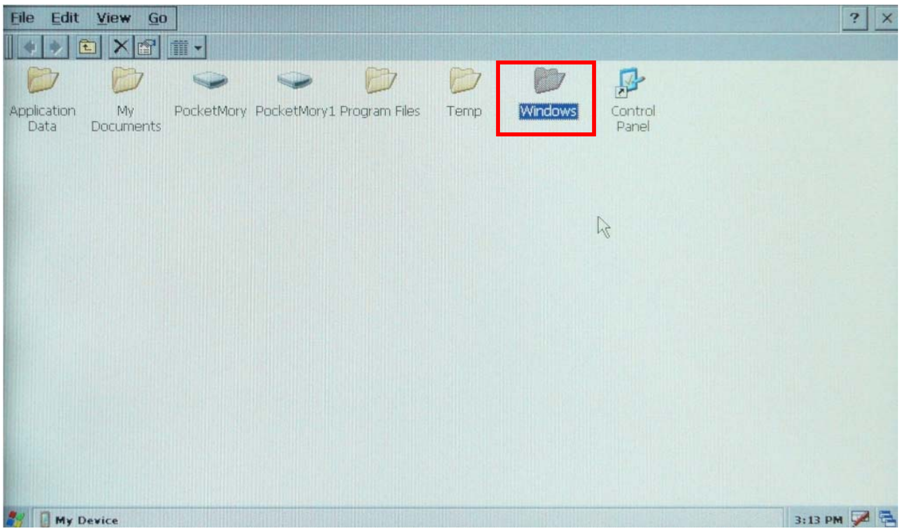
CTFPND-5 provides a build-in GPIO test program. Simply follows the recommended GPIO circuit as illustrated on Chapter 3.4.1, then start using this test program to validate the GPIO ports. ◦

Test Procedure:

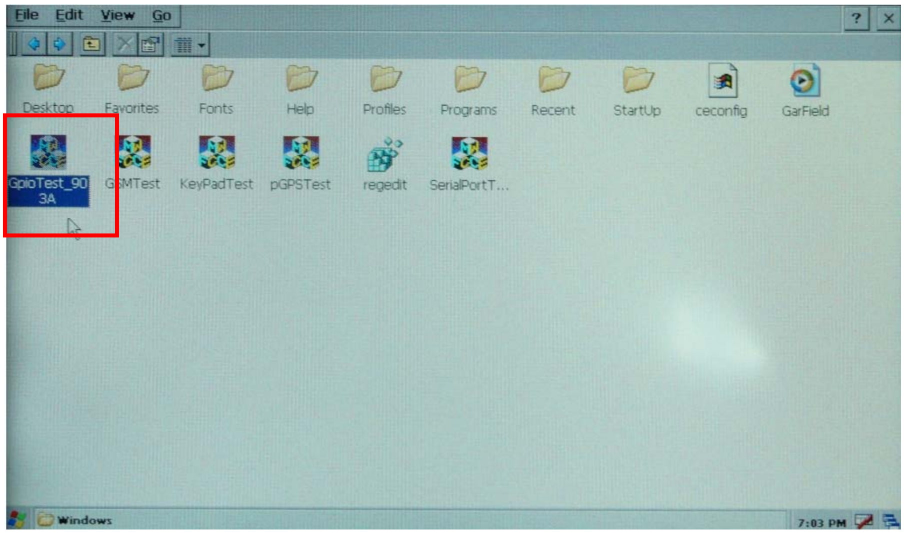
STEP1: On desktop of Windows CE, click on "My Device"



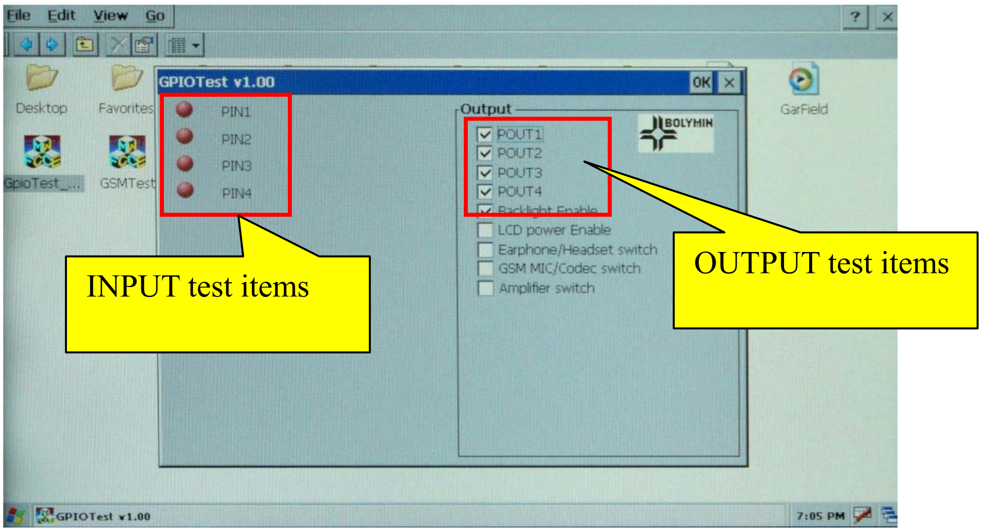
STEP2: select "Windows" folder



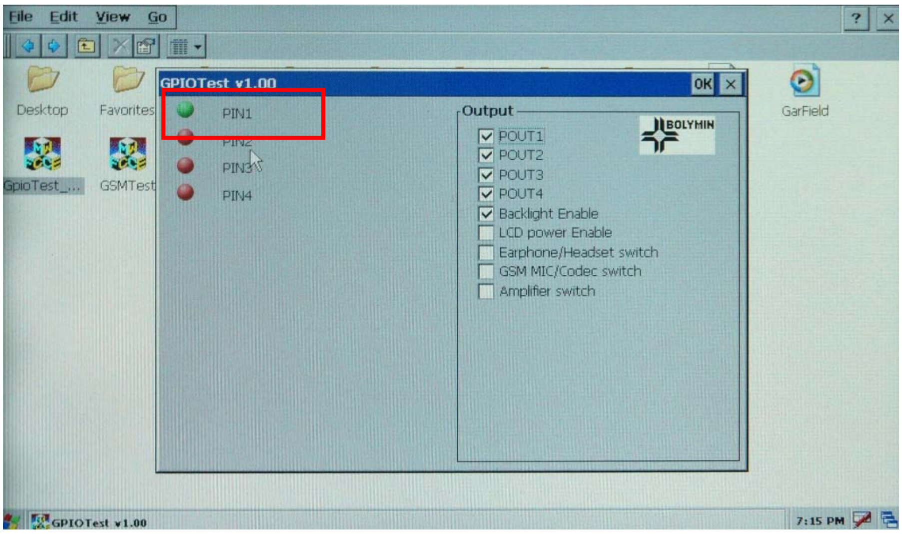
STEP3: Run "GPIO Test_903A" program



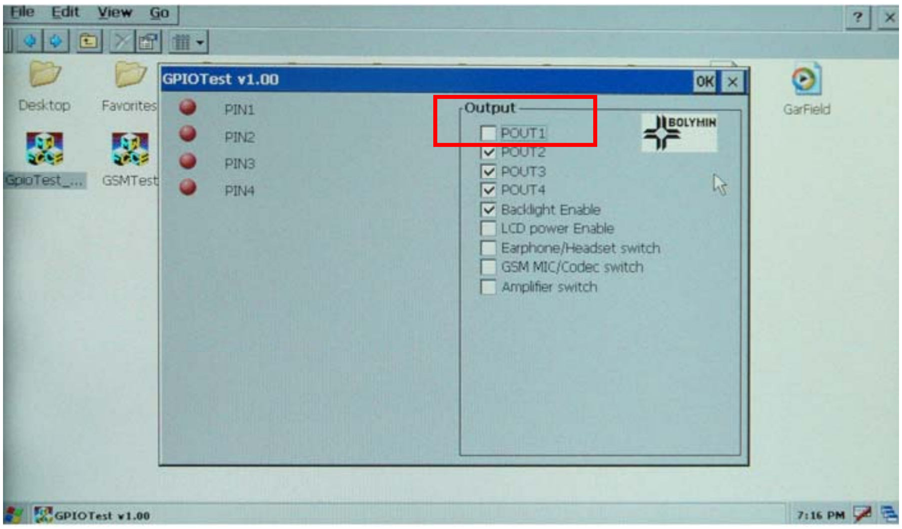
STEP4: A "GPIO Test_903A" running screen



STEP5: PIN1 will be low (red colored) if the switch shunted on IN1 and IN1_GND is open. PIN1 will be high (green colored) if switch is short. 。



STEP6: OUT1 and OUT1_GND will only be active when POUT1 is checked as highlighted ; Inactive if otherwise.

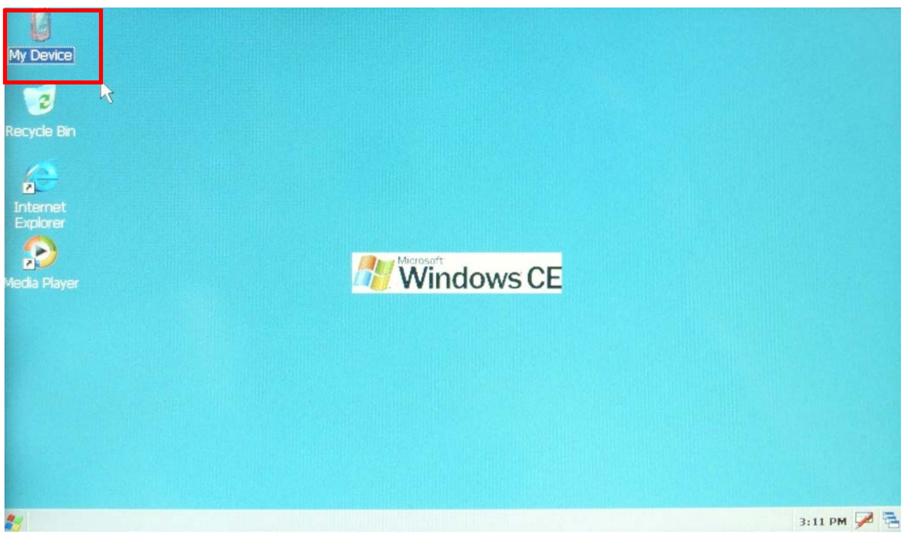


3.5. Keypad Test

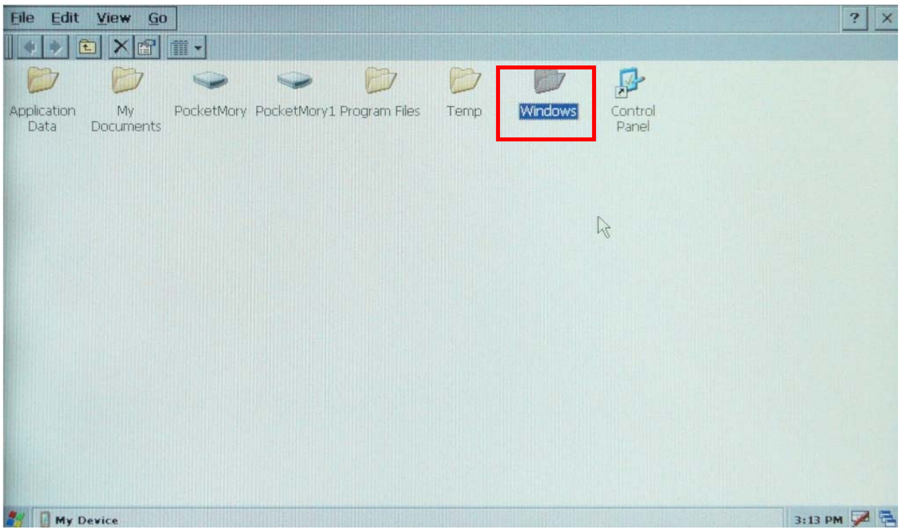
CTFPND-5 supports 8 custom-defined keypads, usu. defined as menu buttons.

Test Procedure:

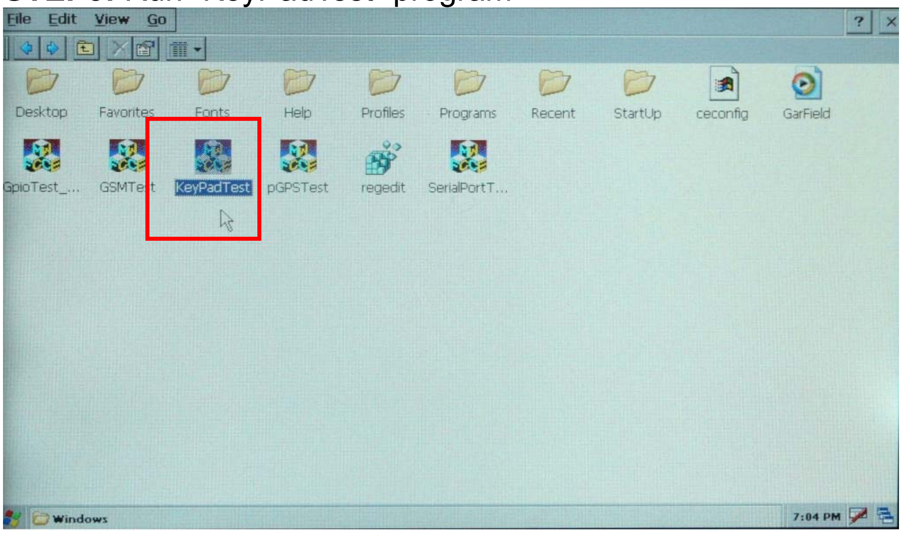
STEP1: On desktop of Windows CE, click on "My Device"



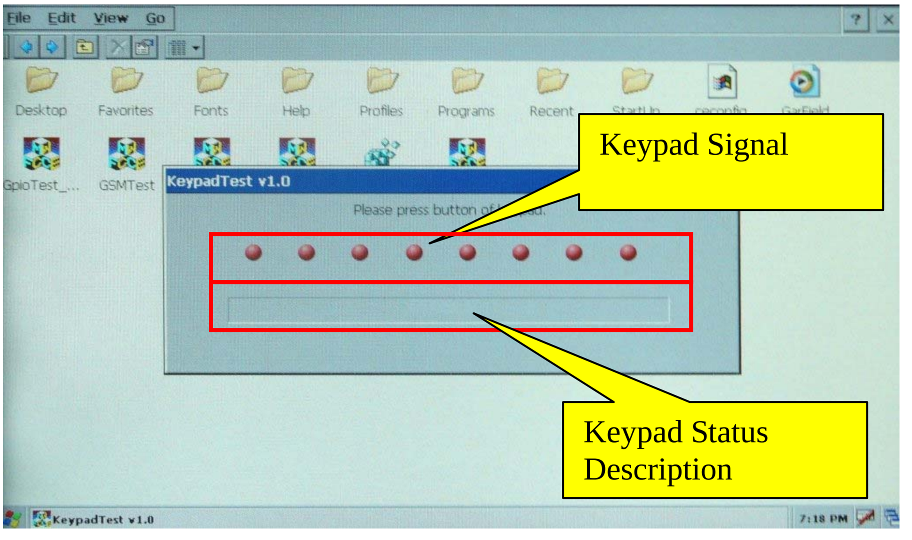
STEP2: Select “Windows” folder



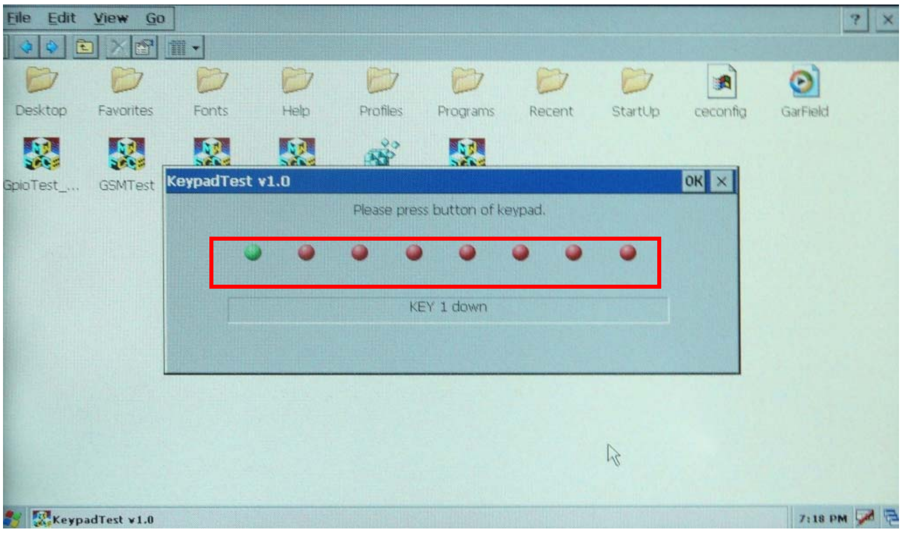
STEP3: Run ”KeyPadTest” program



STEP4: A "KeyPadTest" running screen.



STEP5: Keypad test program will indicate green light or high level when keypad is pressed;red light or low level if otherwise.



4. CTFPND-5 Programming Guide

This chapter provides a kick-start for embedded programming. Some preliminary examples help designers to get a feel about the development environment using Microsoft Embedded Visual C++ 4.0. Sample program demonstrates how to control GPS , GSM , GPIO, Keypad, and Serial Port through software control.

4.1. File Between CTFPND-5 And PC

4.1.1. Connect PC and CTFPND-5

Here is a simple 3 steps to establish a connection between desktop PC and CTFPND-5 :

STEP 1. Install Microsoft ActiveSync 4.5 on desktop PC. You may download the ActiveSync program from here:

http://www.microsoft.com/downloads/details.aspx?familyid=9e641c34-6f7f-404d-a04b-dc09f8141141&displaylang=en&tm

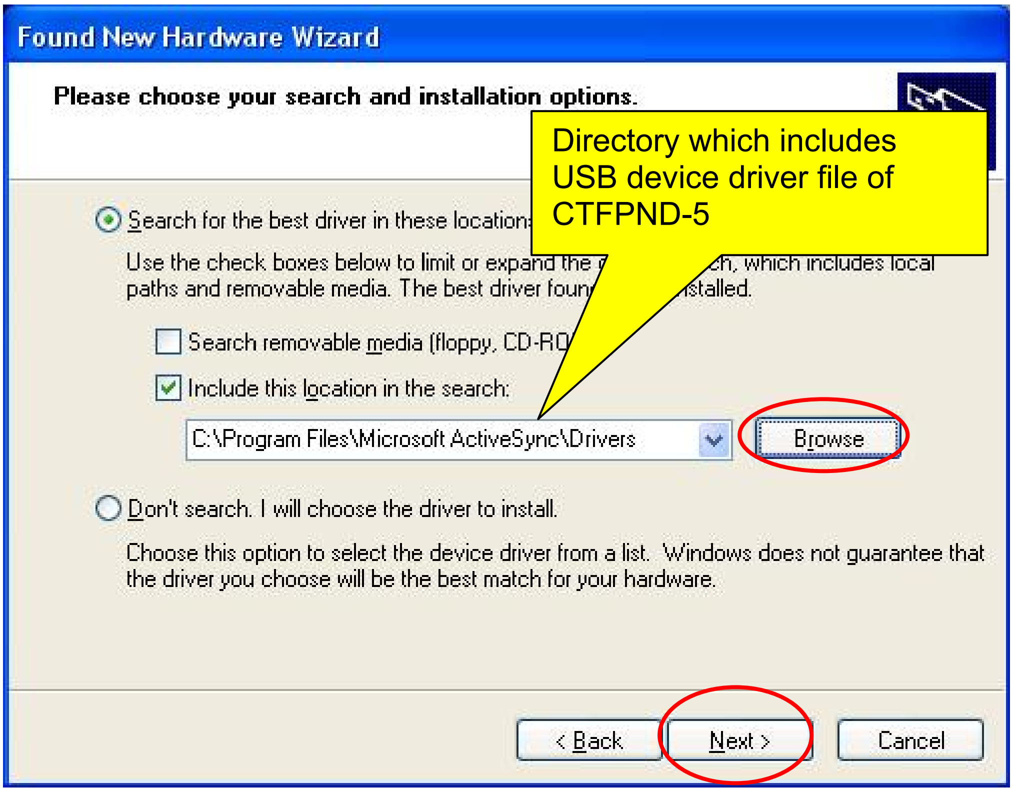
After installation, you need to restart PC.

STEP 2. Connect desktop PC and CTFPND-5 by USB cable. The following picture shows the hardware connection between desktop PC and CTFPND-5 for file transfer.

STEP 3. Connect CTFPND-5 to PC through a USB cable, then power on CTFPND-5. For the first-time connection, windows system on PC will request for the USB device driver of CTFPND-5 Please install USB driver as follows:

(1). Select the advance item on below dialog and click “Next” button.

(2). Click “Browse” button and then select the directory which includes USB device driver file of CTF-PND-5. Click “Next” button.



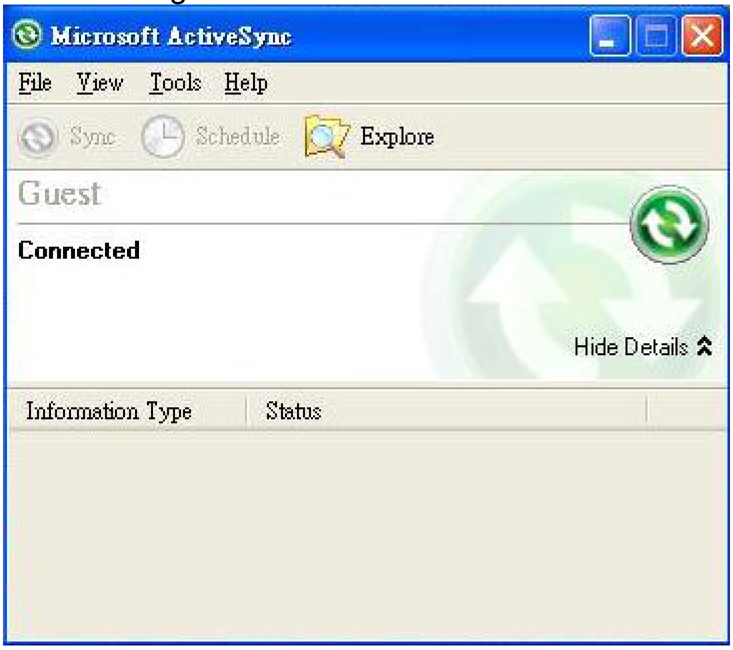
(3). Click “Continue” button

(4). Click “Finish” button. Now CTFPND-5 will connect to PC by ActiveSync.

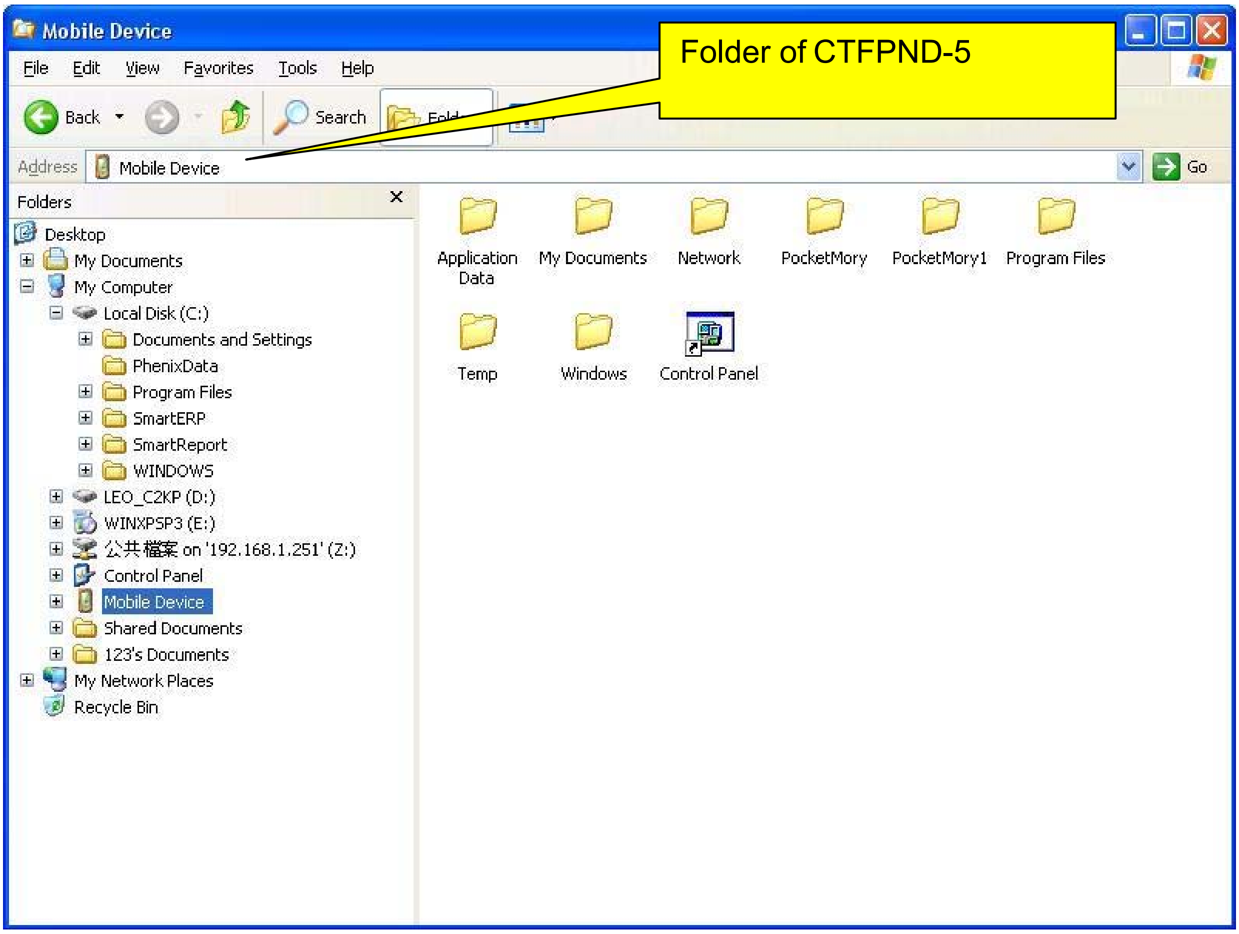
(5). Select “No” and click “Next” button to cancel the synchronization.

4.1.2. Transfer File

Once the USB connection is established, your ActiveSync application will show a "connected" status. The green circle means the connection between PC and CTFPND-5 is done sucessfully.



Run "Explore" program and browse into the folder of CTFPND-5. You can easily transfer files between PC and CTFPND-5



4.2. Programming For CTFPND-5

4.2.1. Set up Development environment

You may set up the WinCE5.0 development environment by following steps:

1. Install Microsoft eMbedded Visual C++ 4.0(eVC 4.0) into desktop PC : eVC 4.0 can be downloaded from

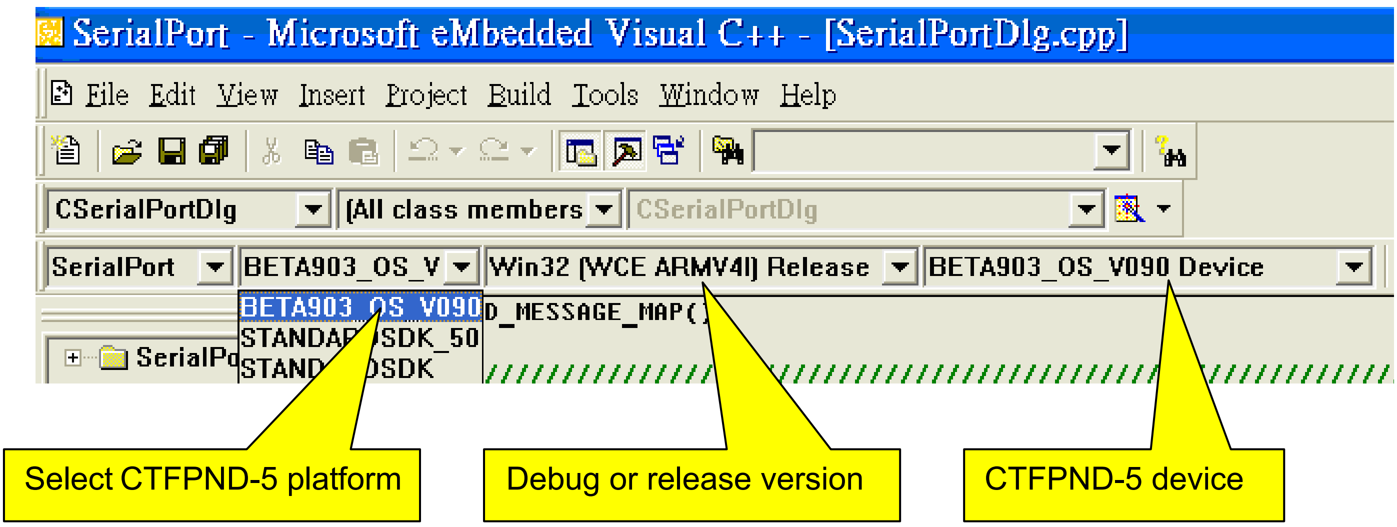
http://www.microsoft.com/downloads/details.aspx?FamilyID=1DACDB3D-50D1-41B2-A107-FA75AE960856&displayLang=en.

2. Connect CTFPND-5 and Desktop PC by procedures in section 4.1.

3. **Install SDK of CTFPND-5.** The installation file could be found in the product CD.

4. The platform setting of embedded Visual C++:

Following pictures show the necessary setting of eVC 4.0::



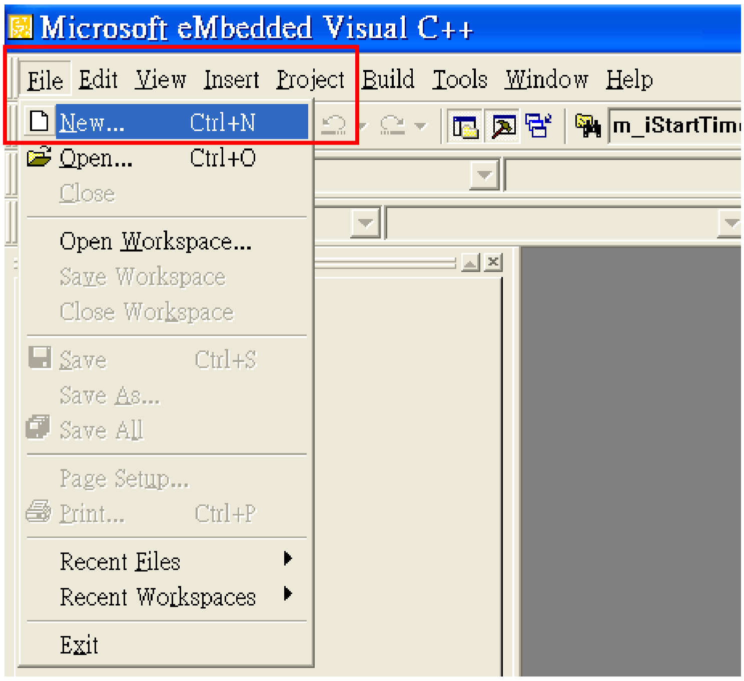
4.2.2. Create New Project

In this section, we will show you how to create a new project in eVC 4.0. You may skip this section if you are already familiar with WinCE development environment.

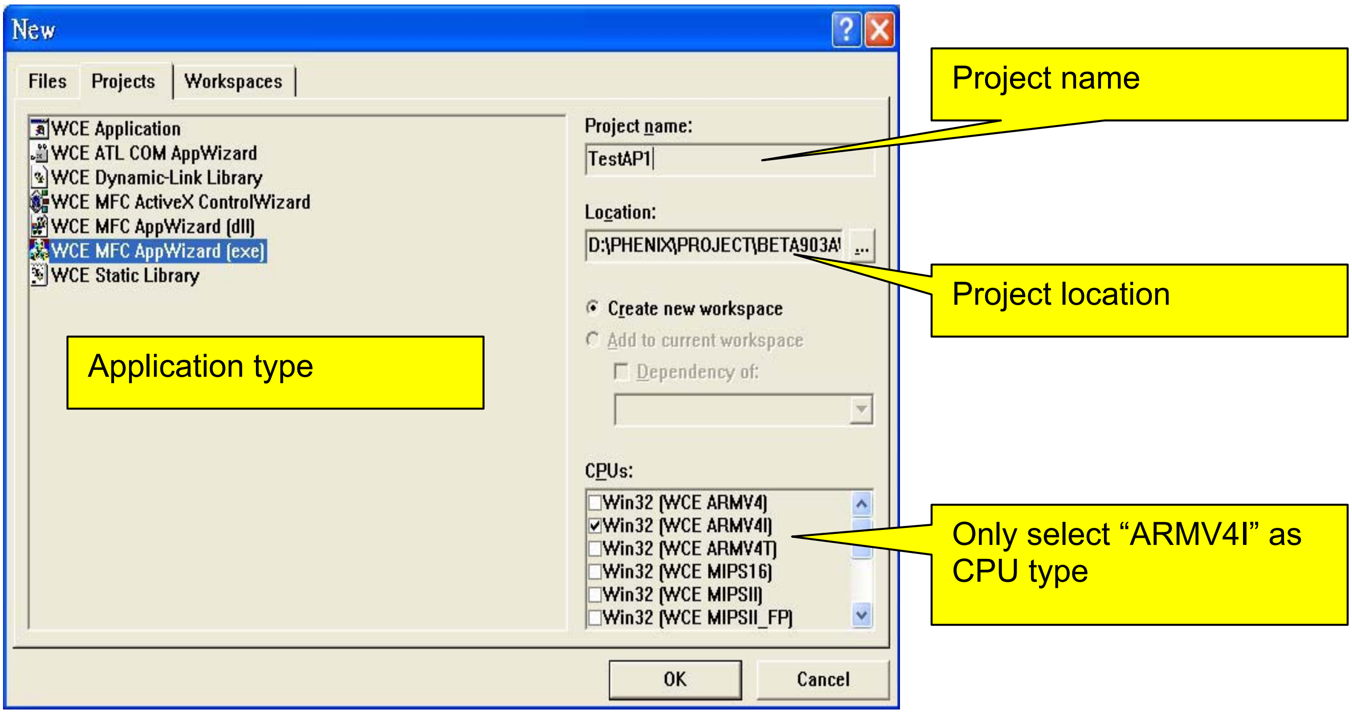
You could create a new project for your application as follows:

STEP 1: Execute eVC 4.0.

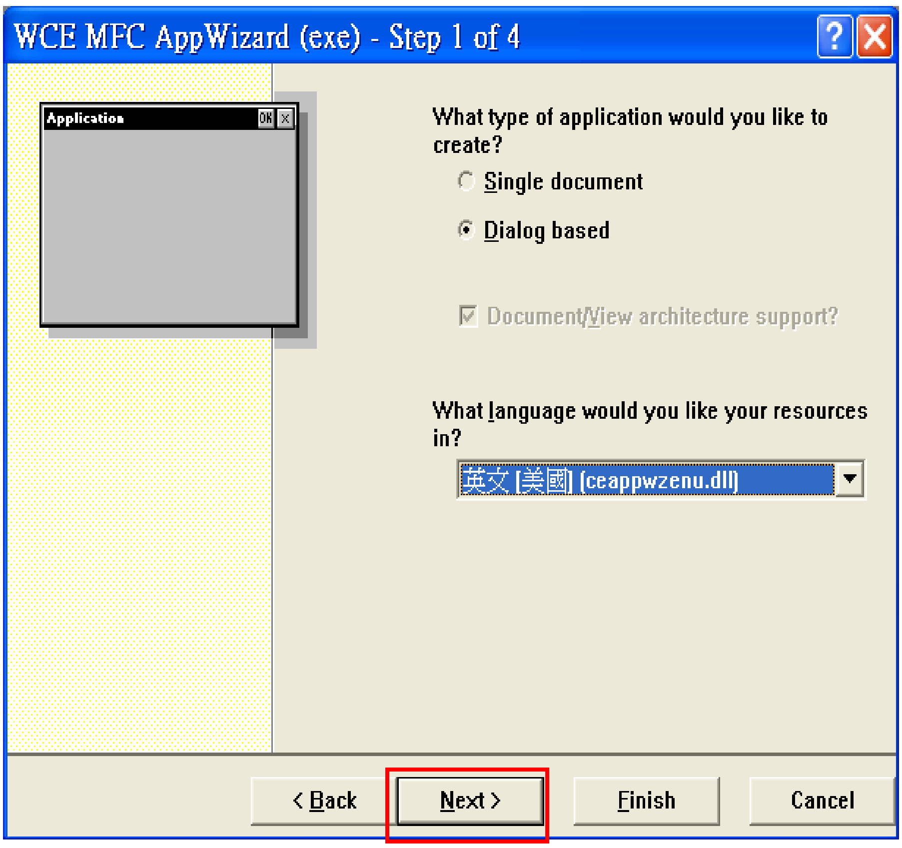
STEP 2: Select “File”-“New…: function



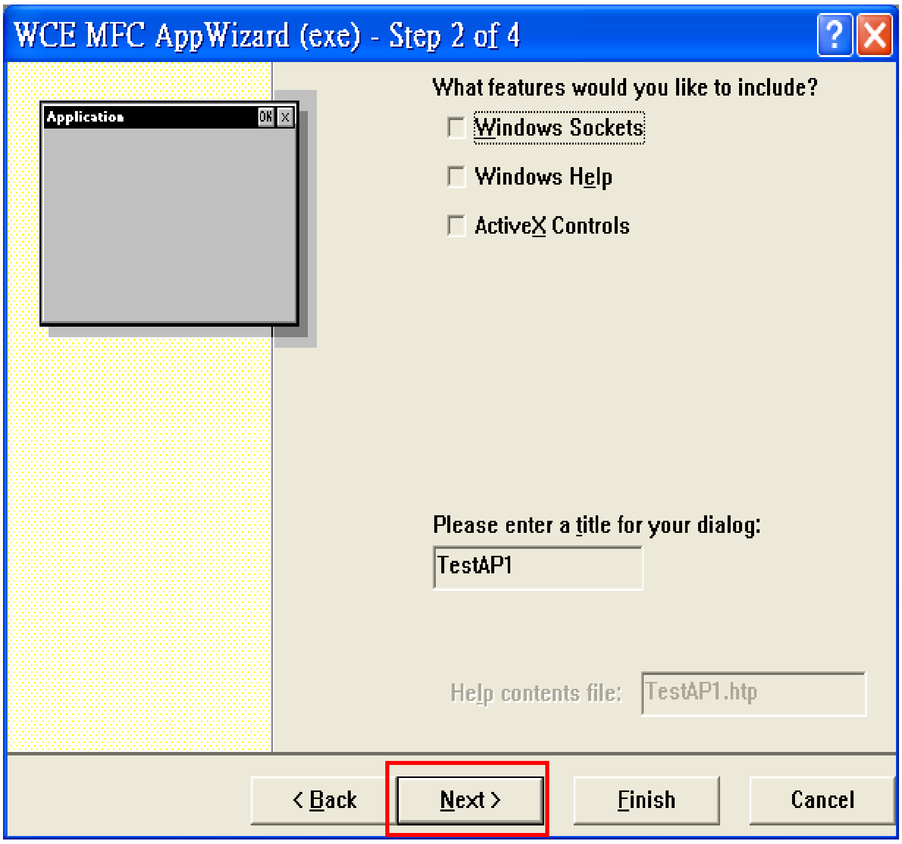
STEP 3: Select your application type, set up the location and name of your project and. Please select "WCE MFC AppWizard(exe)" as application type.



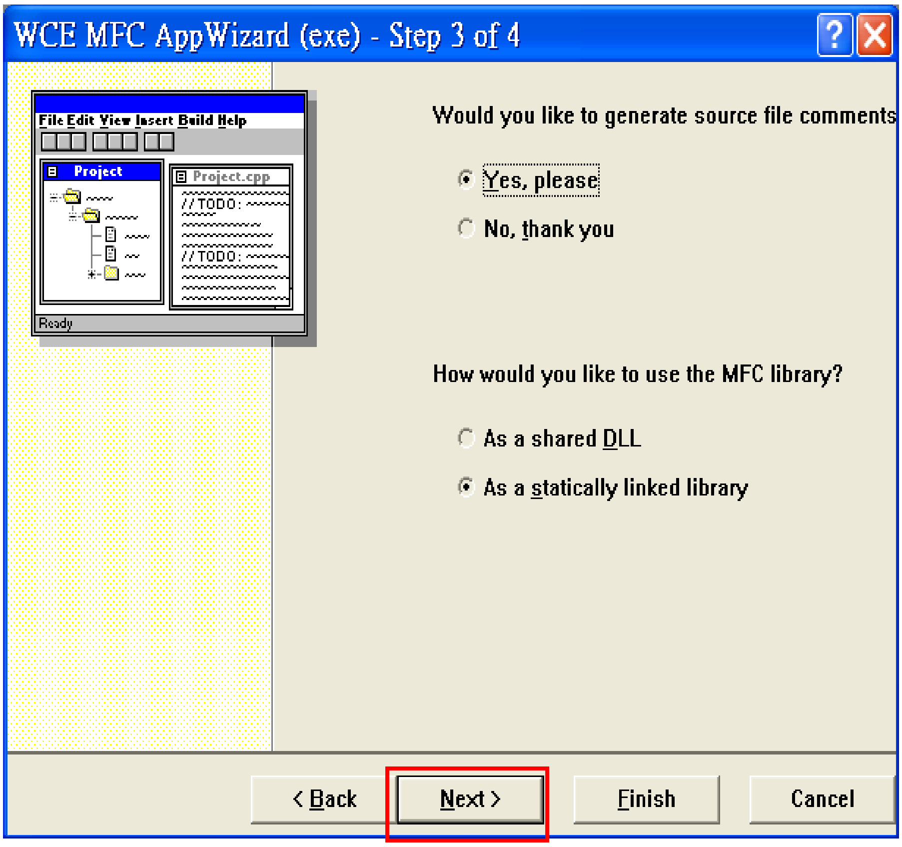
STEP 4: Select "Dialog based" and language setting. Click "Next" button.



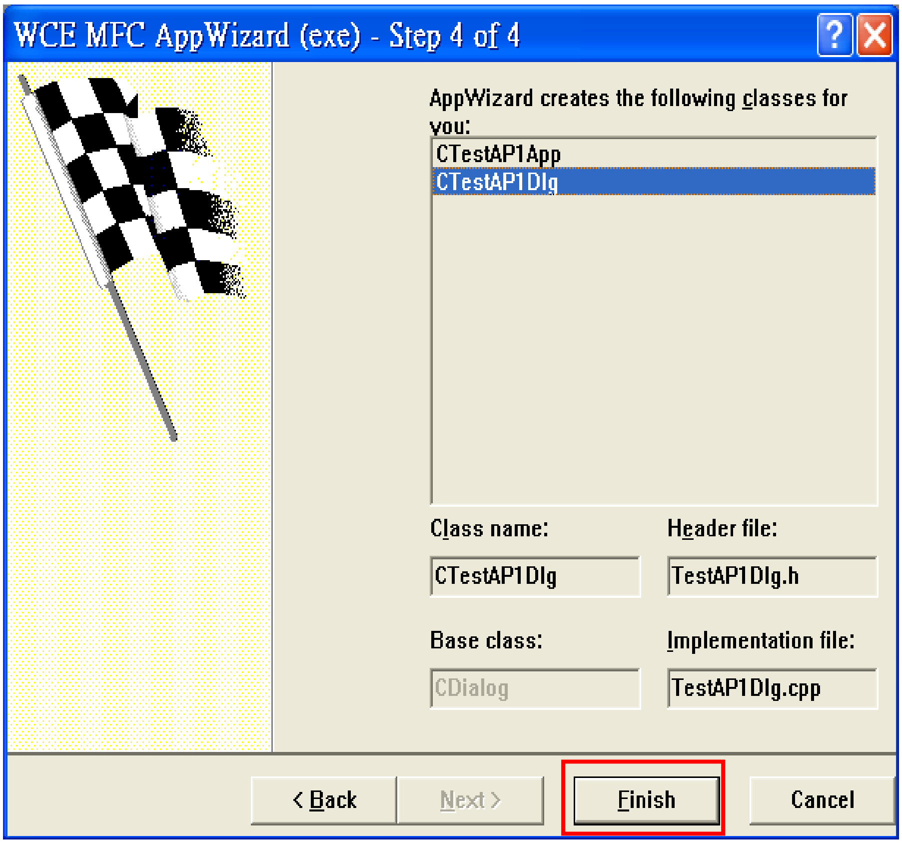
STEP 5: Click "Next" button.



STEP 6: Click "Next" button.



STEP 7: Click “Finish” button.



STEP 8: Now you can add your codes into this new project.
For learn more about WinCE development environment, please explore the MSDN website.
http://msdn.microsoft.com/en-us/library/bb847963.aspx

4.3. Serial Port Function

4.3.1. Overview

CTFPND-5 supports 5 serial ports. And through these serial port, designer may control GSM, GPS and internal debug port. Below table lists the function of each serial port:

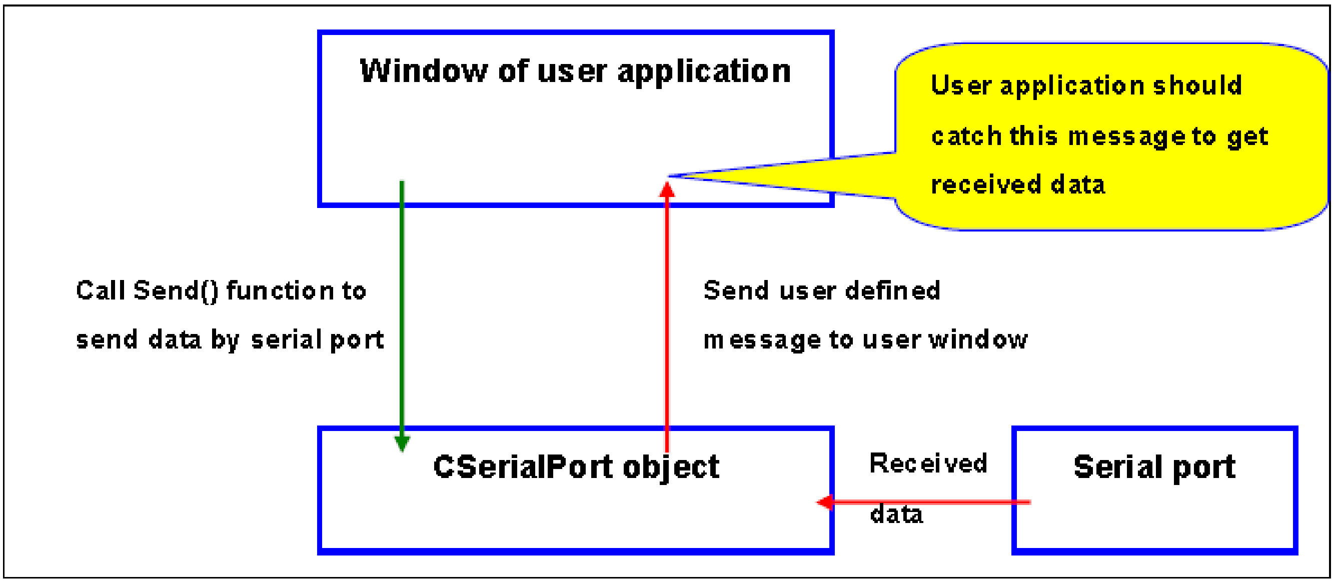
Name	Function	Comment
COM1:	GSM control port	Detail information could be found in section 4.4
COM4:	GPS control port	Detail information could be found in section 4.5
COM3:	Used by application program.	2-wire RS232
COM6:	Used by application program.	8-wire RS232
Debug port	For internal use only.	Can not be opened by application program.

4.3.2. Serial port control - CSerialPort class

We provided a class, CSerialPort, which implements basic control logic for serial port. Application could use this class by adding **"CSerialPort.cpp"** and **"CSerialPort .h"** into project. Customer could modify the source code of class CSerialPort to expand the serial port functions.

4.3.2.1. Basic concept of class CSerialPort

The CSerialPort object-class will handle all data transfer and receiver of opened serial port. Once any data is received by the opened serial port, CSerialPort object will send a user defined message to user-defined window which should be main window of application program. Here is the data flow diagram:



4.3.2.2. Member functions of class CSerial

CSerialPort Function: Constructor function of class CSerialPort.

Syntax	CSerialPort( )
Parameters	None
Return value	None

Open Function: Open a serial port.

Syntax	BOOL Open(LPCTSTR port, int baud_rate, int data_bit int stop_bit int parity);
Parameters	port Name of serial port listed in the table of section 4.3. baud_rate Baud rate, ex: 9600. data_bit Data_bit, 7 ~ 8 stop_bit Stop bit , ONESTOPBIT, ONE5STOPBITS or TWOSTOPBITS. parity Parity , NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY.
Return value	TRUE: Open serial port successfully FALSE: .Open serial port fail.

Send Function: Send specific data by this serial port.

Syntax	BOOL Send(LPCVOID buf_ptr, DWORD data_len);
Parameters	buf_ptr Memory pointer of data will be sent. data_ len Length of data will be sent. (UNIT: byte)
Return value	TRUE: Send data successful. FALSE: Send data fail.

SetCommMsg Function: CSerialPort object will send a receiving message to specified window. User need to call this function to set the receiving message value and the window that will receive message.

Syntax	void SetCommMsg(HWND win_handle, UINT receive_msg);
Parameters	win_handle Handle of the window that will receive message. receive_msg User defined message value.
Return value	None

Close Function: Close current serial port.

Syntax	BOOL Close ( );
Parameters	None
Return value	TRUE: Close serial port successfully. FALSE: Cloas serial port fail.

4.3.2.3. How to catch the receive message

You may catch the received message as follows:

STEP 1: Define a receive message in your code as below:
const UINT WM_CMD_OK = WM_USER+1;

STEP 2: Declare a message processing function in the window that will process receive message.

```
// Generated message map functions
//{{AFX_MSG(CSerialPortDlg)
virtual BOOL OnInitDialog();
afx_msg void OnOpenCom();
afx_msg void OnCloseCom();
afx_msg void OnSend();
afx_msg void OnClearSend();
afx_msg void OnClearRec();
afx_msg void OnDestroy();
afx_msg void OnCmdTest();
//}}AFX_MSG
afx_msg LRESULT OnCommRecv(WPARAM wParam, LPARAM lParam);
DECLARE_MESSAGE_MAP()
```

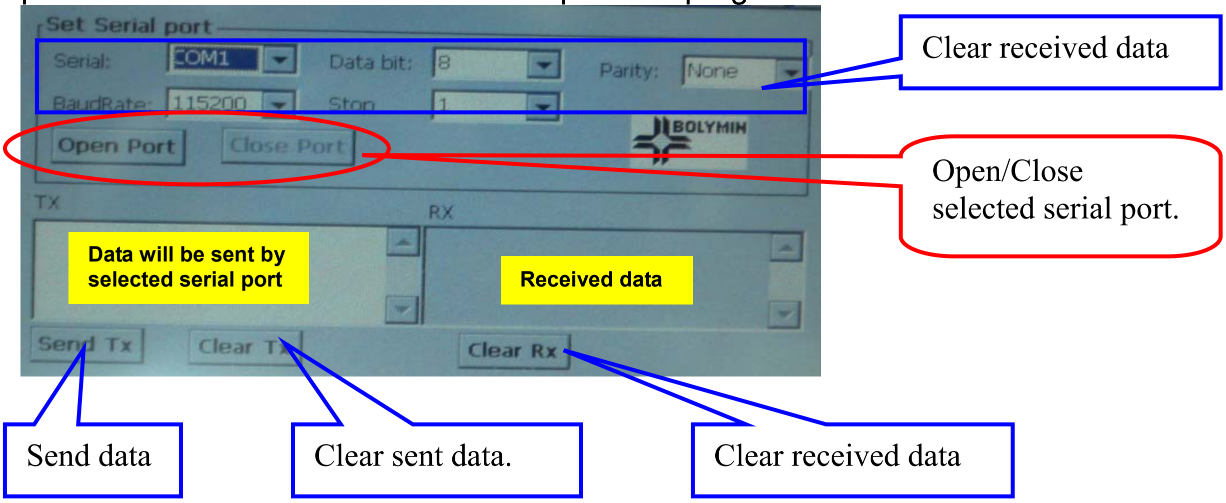
STEP 3: Create message mapping.

```
BEGIN_MESSAGE_MAP(CSerialPortDlg, CDialog)
    //{{AFX_MSG_MAP(CSerialPortDlg)
    ON_BN_CLICKED(IDC_OPEN_COM, OnOpenCom)
    ON_BN_CLICKED(IDC_CLOSE_COM, OnCloseCom)
    ON_BN_CLICKED(IDC_SEND, OnSend)
    ON_BN_CLICKED(IDC_CLEAR_SEND, OnClearSend)
    ON_BN_CLICKED(IDC_CLEAR_REC, OnClearRec)
    ON_BN_CLICKED(IDC_CMD_TEST, OnCmdTest)
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_CMD_OK, OnCommRecv)
END_MESSAGE_MAP()
```

STEP 4: Implement the receive message processing function.

4.3.3. Example code

We provide a test application and its source code for example.The following picture is the screen shot of the serial port test program:



The major part of source codes of the test program is shown below:

File: SerialPortDlg.cpp

```
/////////////////////////////////////////////////////////////////////////////
// CSerialPortDlg dialog
/////////////////////////////////////////////////////////////////////////////

const UINT WM_CMD_OK = WM_USER+1;
```

Define a receive message

```
BEGIN_MESSAGE_MAP(CSerialPortDlg, CDialog)
    //{{AFX_MSG_MAP(CSerialPortDlg)
    ON_BN_CLICKED(IDC_OPEN_COM, OnOpenCom)
    ON_BN_CLICKED(IDC_CLOSE_COM, OnCloseCom)
    ON_BN_CLICKED(IDC_SEND, OnSend)
    ON_BN_CLICKED(IDC_CLEAR_SEND, OnClearSend)
    ON_BN_CLICKED(IDC_CLEAR_REC, OnClearRec)
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP

ON_MESSAGE(WM_CMD_OK, OnCommRecv)
```

Map the receive message processing function to user defined message.

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSerialPortDlg message handlers
/////////////////////////////////////////////////////////////////////////////
```

```
BOOL CSerialPortDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Set the icon for this dialog. The framework does this automatically
        // when the application's main window is not a dialog
       SetIcon(m_hIcon, TRUE);                        // Set big icon
       SetIcon(m_hIcon, FALSE);                       // Set small icon

       CenterWindow(GetDesktopWindow());       // center to the hpc screen
       m_ComboBaud.SetCurSel(5);               /* Define BaudRate: 115200 */
       m_ComboData.SetCurSel(1);               /* Define data bit: 8 bit */
       m_ComboParity.SetCurSel(0);             /* Define parity: none */
       m_ComboPort.SetCurSel(0);               /* Define searial port: COM1 */
       m_ComboStop.SetCurSel(0);               /* Define stop bit: 1bit */

       m_ButClose.EnableWindow(FALSE);         /* "Close"Button is disable*/
       m_strRecDisp = _T("");
       m_cSendBuffer = new char[60];
       UpdateData(FALSE);

       m_pSerialPort = new CSerialPort();
       m_pSerialPort->SetCommMsg(m_hWnd, WM_CMD_OK);
```

Create a CSerialPort object and set current window as the window which will process received data.

```
       return TRUE;
}
/*******************************************************************************************
Implement function used to process receive data from serial port
*******************************************************************************************/
LRESULT CSerialPortDlg::OnCommRecv(WPARAM wParam, LPARAM lParam)
{
        CString tmp;
        char *buf;
        DWORD buflen;

        buf = (char *)wParam;                   //    memory pointer of received data
        buflen = (DWORD)lParam;                 //    received data length

        CEdit *pRecvStrEdit = (CEdit*)GetDlgItem(IDC_REC_DISP);

        for (int i = 0; i < buflen; i++, buf++)
```

```
  {
      tmp.Format(_T("%c"), *buf);
      m_strRecDisp += tmp;
  }
   pRecvStrEdit->SetWindowText(m_strRecDisp); /* Show */
  return 0;
}

// Initial user interface
const CString PorTbl[4] = {_T("COM1:"),_T("COM3:"),_T("COM4:"),_T("COM6:")};
const DWORD BaudTbl[6] = {4800, 9600, 19200, 38400, 57600,115200};
const DWORD DataBitTbl[2] = {7, 8};
const BYTE StopBitTbl[3] = {ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS};
const BYTE ParityTbl[4] = {NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY};

/*******************************************************************************************
Function for "OPEN" button used to open selected serial port.
*******************************************************************************************/
void CSerialPortDlg::OnOpenCom()
{
        UpdateData(TRUE);

        CString strPort = PorTbl[m_ComboPort.GetCurSel()];
        DWORD baud = BaudTbl[m_ComboBaud.GetCurSel()];
        DWORD databit = DataBitTbl[m_ComboData.GetCurSel()];
        BYTE stopbit = StopBitTbl[m_ComboStop.GetCurSel()];
        BYTE parity = ParityTbl[m_ComboParity.GetCurSel()];

        BOOL ret = m_pSerialPort->Open(strPort, baud, databit, stopbit, parity);
        if (ret == FALSE)
       {
         MessageBox(_T("Open ") + strPort + _T(" Fail!"));
          return;
       }

         m_ButOpen.EnableWindow(FALSE);           /* Disable "open" button */
         m_ButClose.EnableWindow(TRUE);           /* Enable "close" button */
         MessageBox(_T("Open ") + strPort + _T(" is OK!"));
```

Open selected serial port by specified parameter values.

```
   }
/*******************************************************************************************
Function for "CLOSE" button used to close current serial port.
*******************************************************************************************/
void CSerialPortDlg::OnCloseCom()
{
     m_pSerialPort->Close();

     m_ButOpen.EnableWindow(TRUE);            /* Enable "Open" button */
     m_ButClose.EnableWindow(FALSE);          /* Disable "close" button */
}

/*******************************************************************************************
Function for "SEND" button used to send data by serial port.
*******************************************************************************************/
void CSerialPortDlg::OnSend()
{
     UpdateData(TRUE);
     int len = m_strSendEdit.GetLength();
     for(int i = 0; i < len;i++)
     m_cSendBuffer[i] = (char)m_strSendEdit.GetAt(i);;

     BOOL status = m_pSerialPort->Send(m_cSendBuffer, len);
     if (!status)
          MessageBox(_T("Can't write string to COM"),_T("Error"),MB_OK);
 }

/*******************************************************************************************
Destory function of serial port test dialog
*******************************************************************************************/
void CSerialPortDlg::OnDestroy()
{
     CDialog::OnDestroy();

     m_pSerialPort->Close();
     delete m_pSerialPort;
     delete m_cSendBuffer;
}
```

Close current serial port.

Send data by current serial port.

Close current serial port and delete CSerialPort object.

4.4. GSM Control

4.4.1. Overview

User application could communicate with GSM module of CTFPND-5 by **COM1** with the following settings:

Baud rate = 57600, Data bit = 8, Stop bit = 1, No parity.

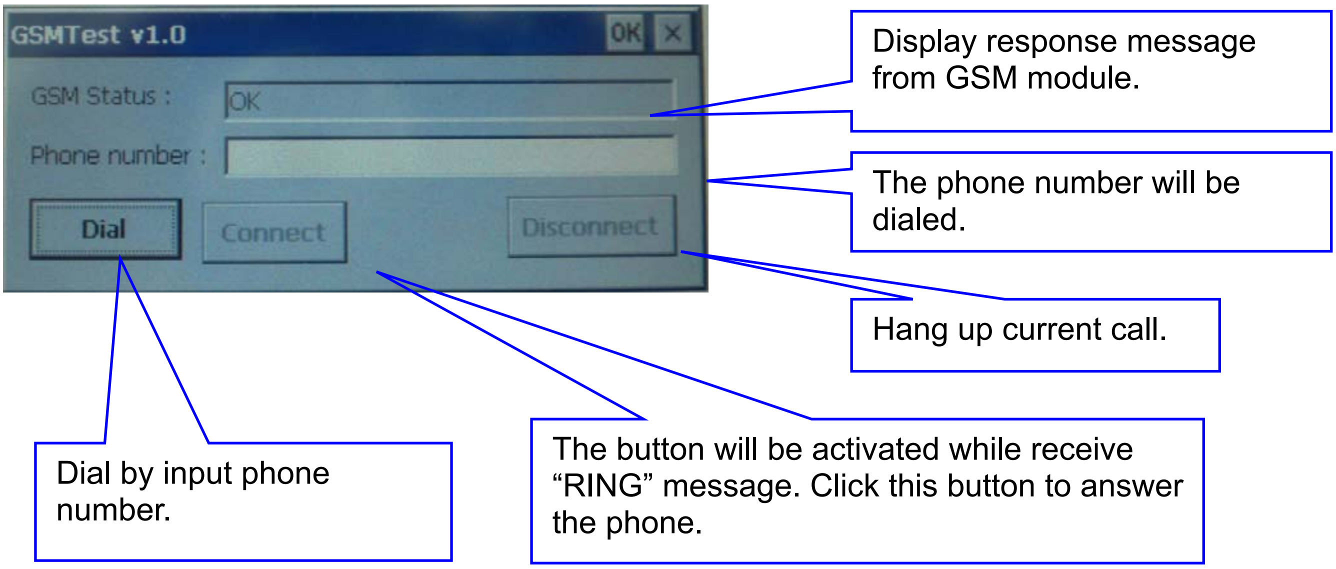
After opening COM1, user could send AT command to control the GSM module. Here is some AT commands used in our example application:

AT command	Description
ATA	Answer a call.
ATH	Disconnect existing call.
ATD<n>	Set up an outgoing call. <n>: phone number.
AT^SNFS=<n>	Set the audio mode required for the connected equipment. **The audio mode should be set to 2 for GSM module of CTFPND-5.**

*More detailed description about AT command could be found at document “MC55i AT Command Set” of Siemens.

4.4.2. Example code

We Provide a simple application and its source code for example. The simple application demonstrates the operations of dial and answer a phone by GSM module. Below picture is the screen shot of the GSM application:



Below are the major source codes of the test program:

File: pGSMTestDlg.cpp

```
const UINT WM_CMD_REVMSG = WM_USER+1;
BEGIN_MESSAGE_MAP(CPGSMTestDlg, CDialog)
	//{{AFX_MSG_MAP(CPGSMTestDlg)
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_BTN_CONNECT, OnBtnConnect)
	ON_BN_CLICKED(IDC_BTN_DISCONNECT, OnBtnDisconnect)
	ON_BN_CLICKED(IDC_BTN_DIAL, OnBtnDial)
	ON_WM_TIMER()
	//}}AFX_MSG_MAP
	ON_MESSAGE(WM_CMD_OK, OnCommRecv)
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////
// CPGSMTestDlg message handlers

BOOL CPGSMTestDlg::SendATCmd(CString cmd)
{
	int len, i;
	len = cmd.GetLength();
	for(i = 0; i < len;i++)
	m_cSendBuffer[i] = (char)cmd.GetAt(i);;
	return (m_pSerialPort->Send(m_cSendBuffer, len));
}
```

Send AT command to GSM module by COM1

```
BOOL CPGSMTestDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	// when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	CenterWindow(GetDesktopWindow());	// center to the hpc screen

	m_cSendBuffer = new char[60];
	m_pSerialPort = new CSerialPort();
m_pSerialPort->SetCommMsg(m_hWnd, WM_CMD_OK);

if (!(m_pSerialPort->Open(_T("COM1:"), 57600, 8, ONESTOPBIT, NOPARITY)))
{
	m_strStatus = _T("Open COM error.");
	m_btnDial.EnableWindow(FALSE);		/* Disable "Dial" button */
	UpdateData(FALSE);

	return TRUE;
}

   CString temp_str;

   // Echo off
   temp_str = _T("ATE0\r");
```

Open COM1 to communicate with GSM module.

Send “Echo off” command to GSM module.

```
    SendATCmd(temp_str);

    m_bSwitchAudioChannel = FALSE;
    m_bConnectCall = FALSE;

    // Load DLL of GPIO control functions of CTFPND-5
    m_hModule=::LoadLibrary(_T("pGPIO_903A.dll"));
    m_pSetGPOutput = (void (*)(int, BOOL))::GetProcAddress(m_hModule,_T("SetGPOutput"));

    return TRUE;     // return TRUE unless you set the focus to a control
}
void CPGSMTestDlg::OnClose()

{
        delete m_pSerialPort;
        delete m_cSendBuffer;

        m_pSetGPOutput = NULL;
        FreeLibrary(m_hModule);

        CDialog::OnClose();
}
/*******************************************************************************************
Implement function used to process receive data from serial port
*******************************************************************************************/
LRESULT CPGSMTestDlg::OnCommRecv(WPARAM wParam, LPARAM lParam)
{
        char *buf;
        DWORD buflen;
        CString tmp;

        buf = (char *)wParam;
        buflen = (DWORD)lParam;

        m_strStatus = _T("");
        for (DWORD i = 0; i < buflen; i++, buf++)
      {
                if (*buf>=0x20)
                {
```

```
                tmp.Format(_T("%c"), *buf);
                m_strStatus += tmp;
            }
    }

    if ((!m_bSwitchAudioChannel)&&(m_strStatus.Find(_T("OK"))!=-1))
    {       // Set audio channel 2
        CString temp_str;

        temp_str = _T("AT^SNFS=2\r");
        SendATCmd(temp_str);
        m_bSwitchAudioChannel = TRUE;
    }
    else if ((m_bConnectCall)&&(m_strStatus.Find(_T("OK"))!=-1))
        SetTimer(1, 200, NULL);          // delay a little time to wait for signal stable(0.2 sec ~ 1.5 sec)
    else if (m_strStatus.Find(_T("RING"))!=-1)
        m_btnConnect.EnableWindow(TRUE);         /* Enable "Connect" button */
    else if (m_strStatus.Find(_T("NO CARRIER"))!=-1)
{
        m_pSetGPOutput(GO_AMP_SWITCH, FALSE);

        m_btnDisConnect.EnableWindow(FALSE);     /* Disable "DisConnect" button */
        m_btnConnect.EnableWindow(FALSE);        /* Disable "Connect" button */
      }

    UpdateData(FALSE);
    return 0;
}

void CPGSMTestDlg::OnBtnConnect()
{
    CString temp_str;

    temp_str = _T("ATA\r");
    SendATCmd(temp_str);

    m_bConnectCall = TRUE;
    m_btnDisConnect.EnableWindow(TRUE);      /* Enable "Connect" button */
}
```

Set the audio mode of GSM module to 2.

Switch amplifier to audio.

Answer incoming call.

```
void CPGSMTestDlg::OnBtnDisconnect()
{
      CString temp_str;

      temp_str = _T("ATH\r");
      SendATCmd(temp_str);

      m_pSetGPOutput(GO_AMP_SWITCH, FALSE);

      m_btnDisConnect.EnableWindow(FALSE);     /* Disable "DisConnect" button */
      m_btnConnect.EnableWindow(FALSE);        /* Disable "Connect" button */
}
```

Hang up current call and switch amplifier to audio.

```
void CPGSMTestDlg::OnBtnDial()
{
      CString temp_str;

      UpdateData(TRUE);

      temp_str.Format(_T("ATD%s;\r"), m_strPhone);
      SendATCmd(temp_str);

      m_bConnectCall = TRUE;



      m_btnDisConnect.EnableWindow(TRUE);      /* Enable "DisConnect" button */
}
```

Setup a call by input phone number.

```
void CPGSMTestDlg::OnTimer(UINT nIDEvent)
{
      m_pSetGPOutput(GO_AMP_SWITCH, TRUE);
      m_bConnectCall = FALSE;

      m_btnConnect.EnableWindow(FALSE);        /* Disable "Connect" button */

      KillTimer(nIDEvent); // Stop the timer

      CDialog::OnTimer(nIDEvent);
}
```

Switch amplifier to GSM function

4.5. GPS Message Translator

The GPS module of CTFPND-5 will send RMC, VTG, GGA, GSA, GASV and GLL messages of NMEA-0183 format from serial port **COM4** with following settings:

Baud rate = 9600, Data bit = 8, Stop bit = 1, No parity.

User application could receive GPS messages by CSerialPort object and translate GPS messages by object of class CGPSTranslator. Application can use this class by adding **"CGPSTranslator.cpp"** and **"CGPSTranslator.h"** into project. CGPSTranslator will translate messages from GPS module into related data structure described in section 4.5.2.

4.5.1. Member functions of class CGPSTranslator

CGPSTranslator Function: Constructor function of calss CGPIO_903A.

Syntax	CGPSTranslator ( );
Parameters	None
Return value	None

Translate Function: GPS message translation function.

Syntax	void Translate (char *msg_buf, int msg_len);
Parameters	msg_buf Memory pointer of GPS message. msg_len length of GPS message.
Return value	None

IsGPSFixed Function: Check if current GPS position fixed..

Syntax	BOOL IsGPSFixed ( );
Parameters	None
Return value	TRUE: GPS position fixed. FALSE: GPS position not fixed.

GetGSVData Function: Get data of the last translated GSV message..

Syntax	void GetGSVData (stGSVData *data_ptr);
Parameters	data_ptr memory pointer of GSV data.
Return value	None

GetVTGData Function: Get data of the last translated VTG message..

Syntax	void GetVTGData (stVTGData *data_ptr);
Parameters	data_ptr memory pointer of VTG data.
Return value	None

GetGSAData Function: Get data of the last translated GSA message..

Syntax	void GetGSAData (stGSAData *data_ptr);
Parameters	data_ptr memory pointer of GSA data.
Return value	None

GetGLLData Function: Get data of the last translated GLL message..

Syntax	void GetGLLData (
	    stGLLData *data_ptr
	  );

Parameters	data_ptr memory pointer of GLL data.
Return value	None

GetGGAData Function: Get data of the last translated GGA message..

Syntax	void GetGGAData (
	    stGGAData *data_ptr
	  );

Parameters	data_ptr memory pointer of GGA data.
Return value	None

GetRMCData Function: Get data of the last translated RMC message..

Syntax	void GetRMCData (
	    stRMCData *data_ptr
	  );

Parameters	data_ptr memory pointer of RMC data.
Return value	None

4.5.2. Data structure of GPS data

Data structure: stRMCData

Data type	Data name	Description
char	cUTCTime[10]	UTC time of fix by hhmmss.ss.
char	cDataStatus	Data status. (A=valid position, V= navigation receiver warning)
double	dLatitude	Latitude of fix.
char	cLatitudeNS	N or S of Latitude.
double	dLongitude	Longitude of fix.
char	cLongitudeEW	E or W of longitude
double	dSpeedInKnots	Speed over ground in knots.
double	dTrackInDegree	Track made good in degrees True.
char	cUTCDate[8]	UTC date of fix by ddmmyy.
double	dMagneticDegrees	Magnetic variation degrees.
char	cMagneticEW	E or W of magnetic variation.
char	cMode	Mode indicator. (A=Autonomous, D=Differential, E=Estimated, N=Data not valid)

Data structure: stVTGData

Data type	Data name	Description
double	dCourseDegree	True course made good over ground by degree.
char	cCourseIndicator	Course indicator.
double	dMagneticDegree	Magnetic course made good over ground by degrees.
char	cMagneticIndicator	Magnetic indicator.
double	dGroundSpeedInKnots	Speed over ground in knots.
char	cGroundSpeedUintKnot	Unit of previous field, N=Knots.
double	dGroundSpeedInKM	Speed over ground in km/hour.
char	cGroundSpeedUintKM	Unit of previous field, K=Kilometers per hour.
char	cMode	Mode indicator. (A=Autonomous, D=Differential, E=Estimated, N=Data not valid)

Data structure: stGGAData

Data type	Data name	Description
char	cUTCTime[10]	UTC time of fix by hhmmss.ss.
double	dLatitude	Latitude of fix.
char	cLatitudeNS	N or S of Latitude.
double	dLongitude	Longitude of fix.
char	cLongitudeEW	E or W of longitude

char	cFixQuality	Fix Quality. (0 = Invalid, 1 = GPS fix, 2 = DGPS fix 1 Data is from a GPS fix)
Int	iSVCount	Number of Satellites in view.
double	dHDOP	Horizontal Dilution of Precision (HDOP).
double	dAltitude	Altitude above mean sea level.
char	cAltitudeUint	Unit of previous field. M=meter.
double	dHOG	Height of geoid above WGS84 ellipsoid.
char	cHOGUint	Unit of previous field. M=meter.

Data structure: stGLLData

Data type	Data name	Description
double	dLatitude	Latitude of fix.
char	cLatitudeNS	N or S of Latitude.
double	dLongitude	Longitude of fix.
char	cLongitudeEW	E or W of longitude
char	cFixTime[10]	UTC time of fix by hhmmss.ss.
char	cDataValid	Data status. (A=valid data)

Data structure: stGSAData

Data type	Data name	Description
char	cMode	Mode. (M=Manual, forced to operate in 2D or 3D A=Automatic, 3D/2D)
int	iModeValue	Mode value. (1=Fix not available, 2=2D, 3=3D)
int	iPRN[12]	PRN of Satellite Vechicles(SV's) used in position fix. (0 for unused field.)
double	dPDOP	Position Dilution of Precision (PDOP)
double	dHDOP	Horizontal Dilution of Precision (HDOP)
double	dVDOP	Vertical Dilution of Precision (VDOP)

Data structure: stGSVData

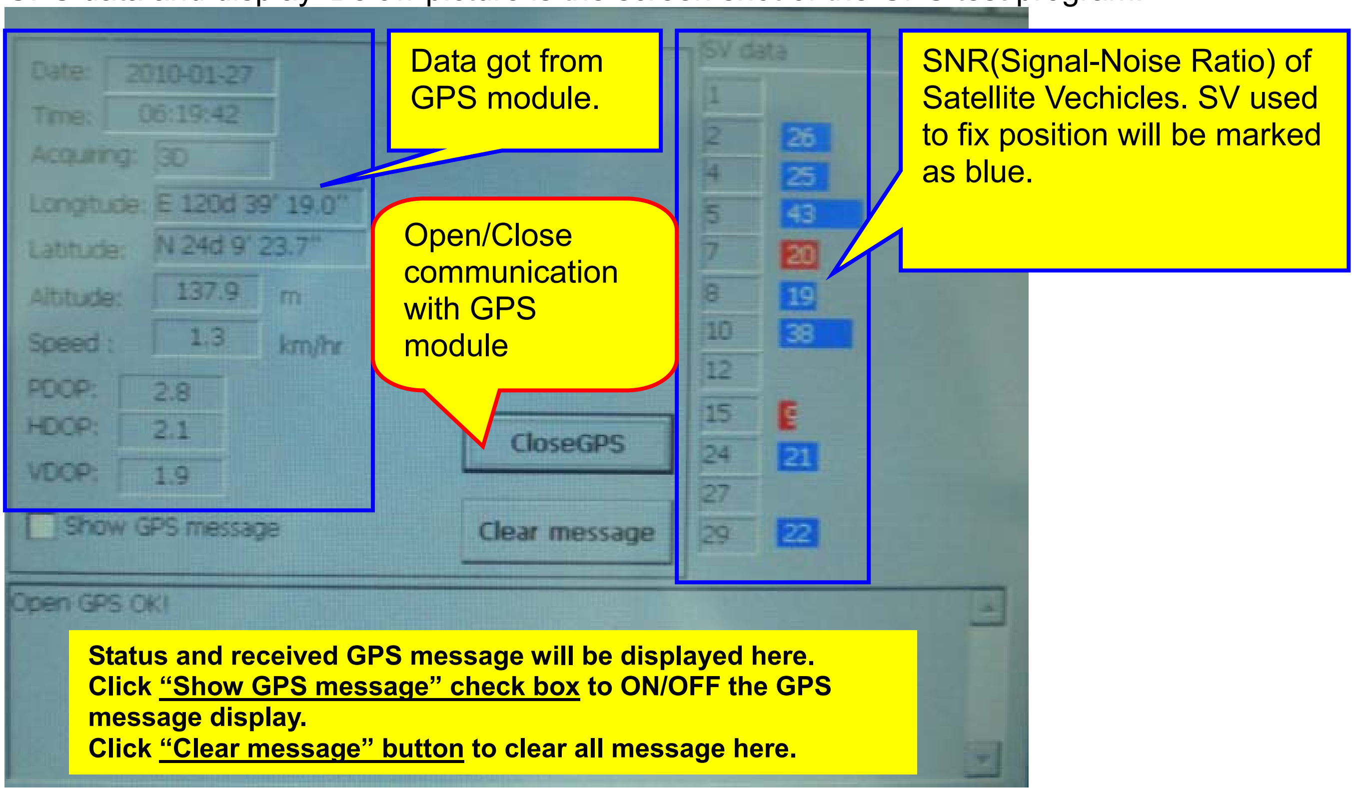
Data type	Data name	Description
int	iSVCount	Number of Satellites in view.
stSVData	SVData[12]	Data of Satellites in view.

Data structure: stSVData

Data type	Data name	Description
int	iPRN	SV PRN number.
int	iElevInDegree	Elevation in degrees. 90 maximum.
int	iAziInDegree	Azimuth degrees from true north, 000~359.
int	iSNR	Signal-Noise Ratio, 00~99 db

4.5.3. Example code

We provide a simple program and its source code for example. The program could receive GPS data and display. Below picture is the screen shot of the GPS test program:



Below are the major source codes of the test program:

File: pGPSTestDlg.cpp

```
const UINT WM_CMD_REVMSG = WM_USER+1;
/////////////////////////////////////////////////////////////////////////////
// CPGPSTestDlg dialog
BEGIN_MESSAGE_MAP(CPGPSTestDlg, CDialog)
	//{{AFX_MSG_MAP(CPGPSTestDlg)
	ON_WM_CLOSE()
	ON_WM_TIMER()
	ON_WM_PAINT()
	ON_BN_CLICKED(IDC_CHECK_SHOWMSG, OnCheckShowmsg)
	ON_BN_CLICKED(IDC_BTN_CLEARMSG, OnBtnClearmsg)
	ON_BN_CLICKED(IDC_BTN_OPENGPS, OnBtnOpengps)
	//}}AFX_MSG_MAP
	ON_MESSAGE(WM_CMD_REVMSG, OnCommRecv)
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CPGPSTestDlg message handlers
BOOL CPGPSTestDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog. The framework does this automatically
	// when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE); // Set big icon
	SetIcon(m_hIcon, FALSE); // Set small icon

	CenterWindow(GetDesktopWindow()); // center to the hpc screen

	// TODO: Add extra initialization here
	m_pSerialPort = new CSerialPort();
	m_pSerialPort->SetCommMsg(m_hWnd, WM_CMD_REVMSG);
	m_bOpenGPS = FALSE;

	m_pGPSTranslator = new CGPSTranslator();
	m_iCurBufIndex = 0;
	m_bShowMsg = FALSE;

	m_pstrSV_PRN[0] = &m_strSV_PRN1;
	m_pstrSV_PRN[1] = &m_strSV_PRN2;
	m_pstrSV_PRN[2] = &m_strSV_PRN3;
	m_pstrSV_PRN[3] = &m_strSV_PRN4;
	m_pstrSV_PRN[4] = &m_strSV_PRN5;
	m_pstrSV_PRN[5] = &m_strSV_PRN6;
	m_pstrSV_PRN[6] = &m_strSV_PRN7;
	m_pstrSV_PRN[7] = &m_strSV_PRN8;
	m_pstrSV_PRN[8] = &m_strSV_PRN9;
	m_pstrSV_PRN[9] = &m_strSV_PRN10;
	m_pstrSV_PRN[10] = &m_strSV_PRN11;
	m_pstrSV_PRN[11] = &m_strSV_PRN12;

	m_pstSV_SNR[0] = &m_stSV_SNR1;
	m_pstSV_SNR[1] = &m_stSV_SNR2;
	m_pstSV_SNR[2] = &m_stSV_SNR3;
	m_pstSV_SNR[3] = &m_stSV_SNR4;
	m_pstSV_SNR[4] = &m_stSV_SNR5;
```

Create object of CGPSTranslator.

```
        m_pstSV_SNR[5] = &m_stSV_SNR6;
        m_pstSV_SNR[6] = &m_stSV_SNR7;
        m_pstSV_SNR[7] = &m_stSV_SNR8;
        m_pstSV_SNR[8] = &m_stSV_SNR9;
        m_pstSV_SNR[9] = &m_stSV_SNR10;
        m_pstSV_SNR[10] = &m_stSV_SNR11;
        m_pstSV_SNR[11] = &m_stSV_SNR12;

        return TRUE; // return TRUE unless you set the focus to a control
}

void CPGPSTestDlg::OnBtnOpengps()
{
        if (m_bOpenGPS)
        {       // Close GPS
                if (m_pSerialPort->Close())
                {
                        m_strGPSMsg += _T("Close GPS OK!\r\n");
                        KillTimer(1);
                        m_btnOpenGPS.SetWindowText(_T("OpenGPS"));
                        m_bOpenGPS = !m_bOpenGPS;
                }
                else
                        m_strGPSMsg += _T("Close GPS Fail!\r\n");
        }
        else
        {      // OpenGPS
```

Open COM4 to communicate with GPS module.

```
                if (m_pSerialPort->Open(_T("COM4:"), 9600, 8, ONESTOPBIT, NOPARITY))
                {
                        m_strGPSMsg += _T("Open GPS OK!\r\n");
                        SetTimer(1, 1000, NULL);
                        m_btnOpenGPS.SetWindowText(_T("CloseGPS"));
                        m_bOpenGPS = !m_bOpenGPS;
                }
                else
                        m_strGPSMsg += _T("Open GPS Fail!\r\n");
        }
        UpdateData(FALSE);
}
```

```
void CPGPSTestDlg::OnClose()
{
    if (m_bOpenGPS)
        m_pSerialPort->Close();
    delete m_pSerialPort;
    delete m_pGPSTranslator;

    CDialog::OnClose();
}

/******************************************************************************************
Implement function used to process receive data from serial port
******************************************************************************************/
LRESULT CPGPSTestDlg::OnCommRecv(WPARAM wParam, LPARAM lParam)
{
     CString tmp;
     char *buf;
     DWORD buflen;
     int i;

     buf = (char *)wParam;
     buflen = (DWORD)lParam;

     if (m_bShowMsg)
     {
          for (i = 0; i < buflen; i++, buf++)
          {
               tmp.Format(_T("%c"), *buf);
               m_strGPSMsg += tmp;
           }

          UpdateData(FALSE);
      }

// Catch a complete GPS message. Start with '$', end by '*', ignore checksum.
i = 0;
while (buflen>0)
{
     if (m_iCurBufIndex==0)
     {    //     message start, find '$'
```

```
                while ((buf[i] != '$')&&(buflen>0))
                {
                        i++;
                        buflen--;
                }
         }

        if (buflen>0)
        {
                m_cMsgBuf[m_iCurBufIndex] = buf[i];
                m_iCurBufIndex++;
                if (buf[i]=='*')
                {   //    Send to translator
                        m_pGPSTranslator->Translate(&m_cMsgBuf[0], m_iCurBufIndex);
                        m_iCurBufIndex = 0;
                }
                i++;
                buflen--;
          }
  }
  return 0;
}

void CPGPSTestDlg::OnTimer(UINT nIDEvent)
{
        UpdateScreen();

        CDialog::OnTimer(nIDEvent);
}
/*******************************************************************************************
Implement function used to display current GPS data
*******************************************************************************************/
void CPGPSTestDlg::UpdateScreen()
{
    stRMCData data_RMC;
    stGSAData data_GSA;
    stGGAData data_GGA;
    stVTGData data_VTG;
    stGSVData data_GSV;
```

Send current received message to GPSTranslator

Variables used to get current GPS data.

```
	int tmp_int[3];
	double tmp_double;
```

Get current RMC data

```
m_pGPSTranslator->GetRMCData(&data_RMC);
m_strDate.Format(_T("20%c%c-%c%c-%c%c"),data_RMC.cUTCDate[4], data_RMC.cUTCDate[5],
data_RMC.cUTCDate[2], data_RMC.cUTCDate[3], data_RMC.cUTCDate[0], data_RMC.cUTCDate[1]);
m_strTime.Format(_T("%c%c:%c%c:%c%c"),data_RMC.cUTCTime[0], data_RMC.cUTCTime[1],
data_RMC.cUTCTime[2], data_RMC.cUTCTime[3], data_RMC.cUTCTime[4], data_RMC.cUTCTime[5]);

if (data_RMC.cLatitudeNS == 0)
{	// Not fixed, initial all fields
	m_strAcquire = _T("No Fix");
	m_strLongitude.Format(_T("%c %dd %d' %.1f\""), ' ', 0, 0, 0.0);
	m_strLatitude.Format(_T("%c %dd %d' %.1f\""), ' ', 0, 0, 0.0);
	m_strAltitude.Format(_T("%.1f"), 0.0);
	m_strSpeed.Format(_T("%.1f"), 0.0);
	m_strPDOP.Format(_T("%d"), 0);
	m_strHDOP.Format(_T("%d"), 0);
	m_strVDOP.Format(_T("%d"), 0);
}
else
{
	t mp_int[0] = (int)data_RMC.dLatitude/100;
	tmp_int[1] = (int)data_RMC.dLatitude-tmp_int[0]*100;
	tmp_double = (data_RMC.dLatitude-tmp_int[0]*100-tmp_int[1])*60;
	m_strLatitude.Format(_T("%c %dd %d' %.1f\""), data_RMC.cLatitudeNS, tmp_int[0], tmp_int[1],
	tmp_double);

	tmp_int[0] = (int)data_RMC.dLongitude/100;
	tmp_int[1] = (int)data_RMC.dLongitude-tmp_int[0]*100;
	tmp_double = (data_RMC.dLongitude-tmp_int[0]*100-tmp_int[1])*60;
	m_strLongitude.Format(_T("%c %dd %d' %.1f\""), data_RMC.cLongitudeEW, tmp_int[0], tmp_int[1],
	tmp_double);
```

Get current GSA data

```
	m_pGPSTranslator->GetGSAData(&data_GSA);
	if (data_GSA.iModeValue == 2)
		m_strAcquire = _T("2D");
	else if (data_GSA.iModeValue == 3)
		m_strAcquire = _T("3D");
	else
```

```
        m_strAcquire = _T("Auto");

        m_strPDOP.Format(_T("%.1f"), data_GSA.dPDOP);
        m_strHDOP.Format(_T("%.1f"), data_GSA.dHDOP);
        m_strVDOP.Format(_T("%.1f"), data_GSA.dVDOP);

        m_pGPSTranslator->GetGGAData(&data_GGA);
```

Get current GGA data

```
        m_strAltitude.Format(_T("%.1f"), data_GGA.dAltitude);

        m_pGPSTranslator->GetVTGData(&data_VTG);
```

Get current VTG data

```
        m_strSpeed.Format(_T("%.1f"), data_VTG.dGroundSpeedInKM);
    }

    // Update SV status
    CRect rect;

    m_pGPSTranslator->GetGSVData(&data_GSV);
```

Get current GSV data

```
    for (int i=0; i<12; i++)
    {

        if (data_GSV.SVData[i].iPRN>0)
        {
            m_pstrSV_PRN[i]->Format(_T("%d"), data_GSV.SVData[i].iPRN);
            m_pstSV_SNR[i]->GetWindowRect(&rect);
            ScreenToClient(&rect);
            InvalidateRect(&rect);
        }
        else
            m_pstrSV_PRN[i]->Format(_T("%c"), '-');
    }

    UpdateData(FALSE);
}

// Check if the input SV was used for position fix.
BOOL IsFixedSV(int sv_prn, stGSAData *gsa_data)
{
    for (int i=0; i<12; i++)
    {
        if (gsa_data->iPRN[i] == sv_prn)
            return TRUE;
```

```
    }

    return FALSE;
}

// Paint SNR
void CPGPSTestDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    CRect rect;
    CString tmp_str;
    stGSVData data_GSV;
    stGSAData data_GSA;
    COLORREF bar_color, src_color;

    m_pGPSTranslator->GetGSVData(&data_GSV);
    m_pGPSTranslator->GetGSAData(&data_GSA);
    for (int i=0; i<12; i++)
    {
        m_pstSV_SNR[i]->GetWindowRect(&rect);
        ScreenToClient(&rect);

        if ((data_GSV.SVData[i].iSNR<100)&&(data_GSV.SVData[i].iSNR>0))
            tmp_str.Format(_T(" %d"), data_GSV.SVData[i].iSNR);
        else
            tmp_str = _T(" ");

        rect.right = rect.left+(rect.Width()/100)*data_GSV.SVData[i].iSNR;
        if (IsFixedSV(data_GSV.SVData[i].iPRN, &data_GSA))
            bar_color = RGB(0,0,255);
        else
            bar_color = RGB(255,0,0);
        dc.FillSolidRect(&rect, bar_color);
        src_color = dc.SetBkColor(bar_color);
        dc.SetTextColor(RGB(255, 255, 255));
        dc.DrawText(tmp_str, &rect, 0 );
        dc.SetTextColor(RGB(0, 0, 0));
        dc.SetBkColor(src_color);
    }
}
```

4.6. GPIO And Keypad Control

4.6.1. How to control GPIO

We provides a DLL file “**pGPIO_903A.dll**” to control the General Purpose Input and Ouput(GPIO) signal. User could read current value of all GPIO of CTFPND-5 and change values of GP output signal by functions in “**pGPIO_903A.dll**”.

User could use GPIO control functions by following procedures:

STEP 1. Add **“pGPIO_903A.h”** into project.

STEP 2. Load “**pGPIO_903A.dll**” by “**Loadlibrary**()” function.

STEP 3. Get the address of control functions by “**GetProcAddress**()” function.

STEP 4. Execute GPIO control functions by the address got at STEP3.

Here is a simple example code to use the GPIO control functions:

```
// variable declaration
HINSTANCE m_hModule;
BOOL (*m_pGetGPInput)(int);
void (*m_pSetGPOutput)(int, BOOL);

m_hModule=::LoadLibrary(_T("pGPIO_903A.dll"));
m_pGetGPInput = (BOOL (*)(int))::GetProcAddress(m_hModule,_T("GetGPInput"));
m_pSetGPOutput = (void (*)(int, BOOL))::GetProcAddress(m_hModule,_T("SetGPOutput"));

m_bPOUT1 = m_pGetGPInput(GO_POUT1);
m_pSetGPOutput(GO_BLIGHT_ENABLE, TRUE);
```

Load “**pGPIO_903A.dll**” and get the address of GPIO control functions.

Execute GPIO control functions.

4.6.2. GPIO control functions for CTFPND-5

GetGPInput Function: Get current status of specified GPIO.

Syntax	BOOL GetGPInput (int gpio_index);
Parameters	gpio_index The index of specified GPIO. Refer to section 4.6.3 for the value definition.
Return value	TRUE: Current status of specified GPIO is HIGH. FALSE: Current status of specified GPIO is LOW.

SetGPOutputFunction: Set value of specified GP Output.

Syntax	void SetGPOutput (int gpio_index, BOOL value);
Parameters	gpio_index The index of specified GP output. Refer to section 4.6.3 for the value definition. Value New value of specified GP output. TRUE: Set specified GP output to HIGH. FALSE: Set specified GP output to LOW.
Return value	None

4.6.3. Definition of GPIO index

Class CGPIO_903A supports following index values:

GPIO index	Description
GI_INPUT1	User defined general purpose input. (IN1)
GI_INPUT2	User defined general purpose input. (IN2)
GI_INPUT3	User defined general purpose input. (IN3)
GI_INPUT4	User defined general purpose input. (IN4)
GO_POUT1	User defined general purpose output. (OUT1)
GO_POUT2	User defined general purpose output. (OUT2)
GO_POUT3	User defined general purpose output. (OUT3)
GO_POUT4	User defined general purpose output. (OUT4)
GO_BLIGHT_ENABLE	Backlight control. Default value : **HIGH**.
GO_LCD_POWER_ENABLE	LCD power control. Default value : **LOW**.
GO_EARPHONE_SWITCH	Earphone switch. Default value : **HIGH**. Set LOW when earphone is used for GSM function.
GO_GSM_MIC_SWITCH	Reserved for internal use.
GO_AMP_SWITCH	Amplifier switch. Default vale: **LOW**. Set HIGH when amplifier is used for GSM function.

4.6.4. Keypad control

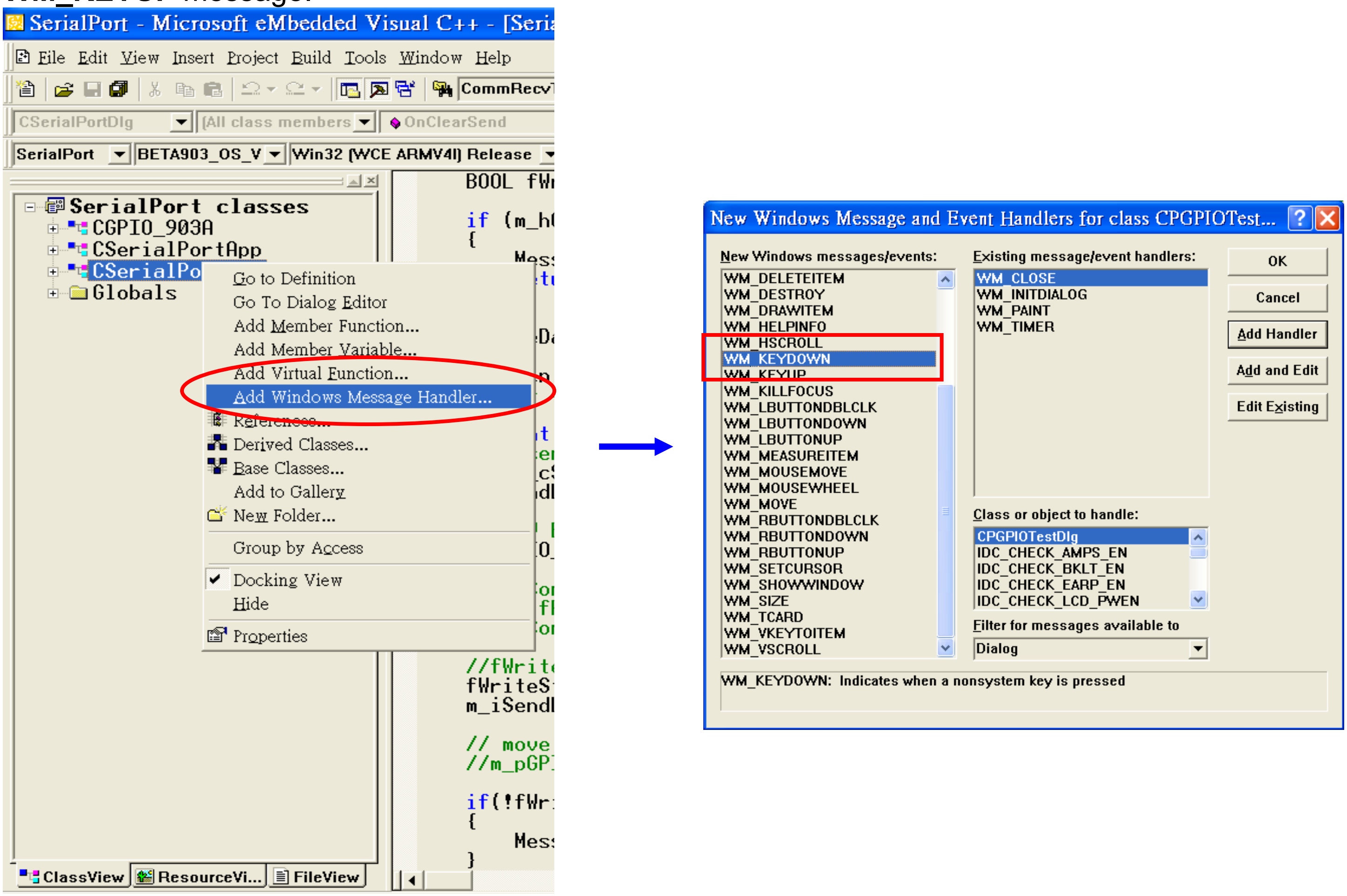
The keypad of CTFPND-5supports 8 user-defined buttons. The buttons of keypad of CTFPND-5 will map to **F11~F18** of keyboard. To learn which button of keypad is pressed, you may catch WM_KEYDOWN or WM_KEYUP message in your program and add the process codes. Here is a step guide:

STEP 1: Override **PreTranslateMessage**() function of the window which will catch key message, as shown below:

```
// Override this function to catch key message
BOOL CPKeypadTestDlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message==WM_KEYDOWN)
    {
        if ((pMsg->wParam>=VK_F11)&&(pMsg->wParam>=VK_F18))
         SendMessage(pMsg->message, pMsg->wParam, pMsg->lParam);
    }

    return CDialog::PreTranslateMessage(pMsg);
}
```

STEP 2: In Class view of eVC 4.0, click right button of mouse on the window that will catch the key message. Select “**Add Windows Message Handler…**” and then select **WM_KEYDOWN** or **WM_KEYUP** message.



STEP 3: Add process code into message processing function.

```
void CPTest1Dlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    switch (nChar)
    {
        case VK_F11:
            m_strEdit = _T("KEY 1 down");
            break;
        case VK_F12:
            m_strEdit = _T("KEY 2 down");
            break;
    }

    UpdateData(FALSE);

    CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

< End of CTFPND-5 User Manual >