

# MCP-903A

## 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





# Table of content

1.	General Information .....	5
1.1.	Introduction .....	6
1.1.1.	Packing contents .....	6
1.1.2.	System outline.....	7
1.2.	Specification .....	12
1.2.1.	System functional blocks .....	12
1.2.2.	System Specifications .....	13
1.2.3.	GPS Module Specifications .....	14
1.2.4.	GPS Antenna Specifications .....	15
1.2.5.	GSM Module specifications .....	16
1.2.6.	GSM antenna specifications .....	17
1.3.	Mechanical specifications .....	18
1.3.1.	Mechanical specifications of module .....	18
1.3.2.	Mechanical specifications of Holder & Mount .....	19
2.	MCP-903A Installation guide.....	20
2.1.	Install a sucker & mount.....	21
2.2.	Install a GSM antenna .....	22
2.3.	Install a SIM card .....	23
2.4.	Install a SD card .....	24
2.5.	Install external GPS antenna .....	25
2.6.	Install battery .....	26
2.7.	Connect Power .....	27
2.8.	MCP-903A power status .....	28
3.	How to test on MCP-903A.....	29
3.1.	Serial Port test .....	30
3.1.1.	Connect Serial Port to PC .....	30
3.1.2.	Serial port test .....	32
3.2.	GSM test.....	37
3.3.	GPS test .....	40
3.4.	GPIO test .....	43
3.4.1.	Recommended interfacing circuit to GPIO .....	43
3.4.2.	GPIO test.....	45
3.5.	Keypad Test .....	48
4.	MCP-903A Programming Guide.....	51

4.1.	File Between MCP-903A And PC .....	52
4.1.1.	Connect PC and MCP-903A .....	52
4.1.2.	Transfer File .....	56
4.2.	Programming For MCP-903A .....	57
4.2.1.	Set up Development environment.....	57
4.2.2.	Create New Project .....	58
4.3.	Serial Port Function.....	62
4.3.1.	Overview .....	62
4.3.2.	Serial port control - CSerialPort class .....	63
4.3.3.	Example code .....	67
4.4.	GSM Control .....	71
4.4.1.	Overview .....	71
4.4.2.	Example code .....	72
4.5.	GPS Message Translator .....	77
4.5.1.	Member functions of class CGPSTranslator .....	78
4.5.2.	Data structure of GPS data .....	80
4.5.3.	Example code .....	82
4.6.	GPIO And Keypad Control .....	90
4.6.1.	How to control GPIO of MCP-903A .....	90
4.6.2.	GPIO control functions for MCP-903A .....	91
4.6.3.	Definition of GPIO index .....	92
4.6.4.	Keypad control.....	93

# 1. General Information

This chapter provides basic information of MCP-903A module and consists of the following:

- 1.1 Introduction
- 1.2 Specifications
- 1.3 Mechanical specifications
- 1.4 Packing contents

# 1.1. Introduction

MCP-903A 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.

MCP-903A 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 MCP-903A, 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 MCP-903A family :

## Order Information

Part No.	Touch Panel	Battery	LCD	Touch Panel
MCP-903A			-30°C~80°C	
MCP-903A1	*		-30°C~80°C	-30°C~80°C
MCP-903A2		*	-30°C~80°C	
MCP-903A3	*	*	-30°C~80°C	-30°C~80°C
MCP-903A4			-20°C~70°C	
MCP-903A5	*		-20°C~70°C	-20°C~60°C
MCP-903A6		*	20°C~70°C	
MCP-903A7	*	*	20°C~70°C	-20°C~60°C

### 1.1.1. Packing contents

Check your package for the following items:

- MCP-903A 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)

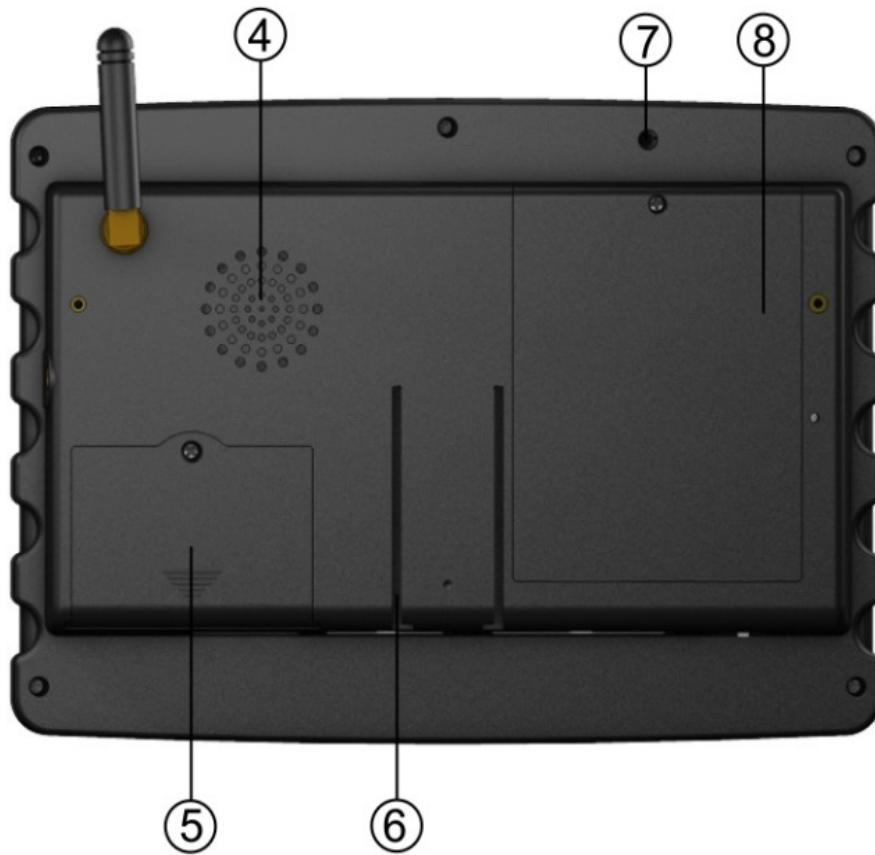
## 1.1.2. System outline



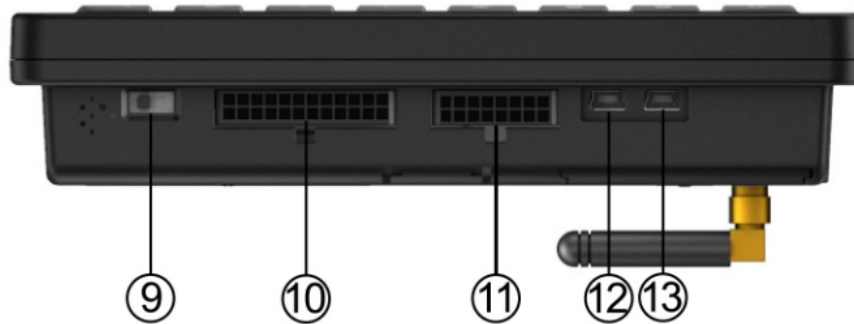
#	Name	Description
1	LED Indicator	Indicates MCP-903 Status
2	Microphone	Supports GSM for voice communication
3	Push Buttons	Supports 8 customized menu buttons (none system default)

### #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



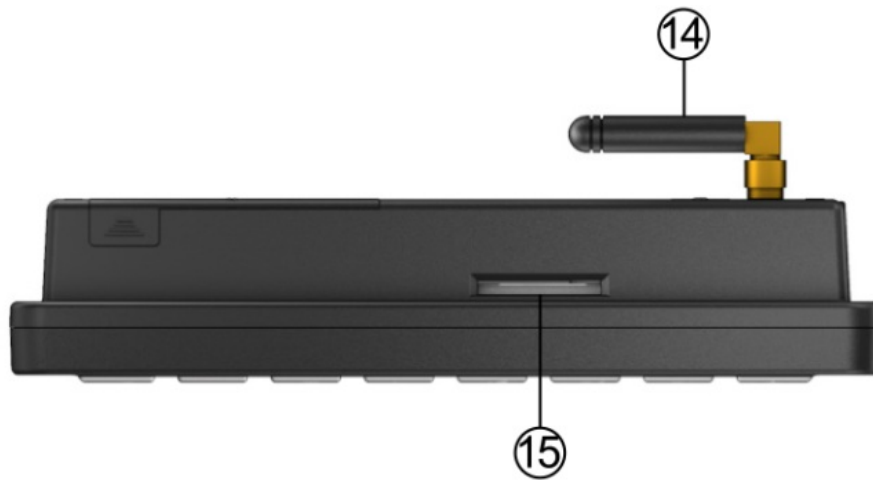
#	Name	Description
9	Power switch	Turn on/off power
10	Power & I/O ports	Power input , 4x photo-coupler input , 4x photo-coupler output
11	Serial I/O	2-wire RS232 x 2, 8-wire RS232 x 1
12	Host –USB	External USB1.0 host for file access
13	Device-USB	Connect to PC for data sync by using <i>ActiveSync</i>

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

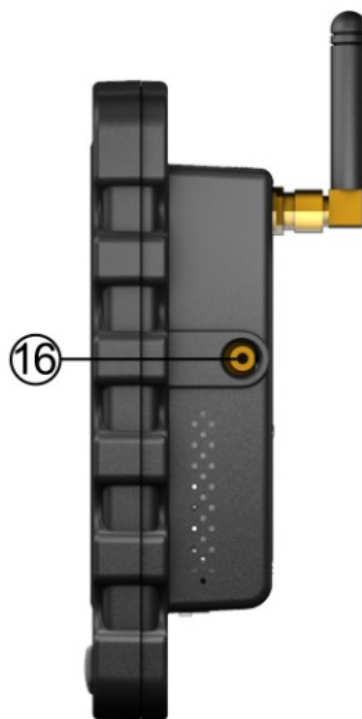
	13	1	13	DCIN	1	DCIN
	14	2	14	GND	2	GND
	15	3	15	NC	3	VIG
	16	4	16	VIO	4	VIO
	17	5	17	IN1	5	OUT1
	18	6	18	IN1_GND	6	OUT1_GND
	19	7	19	IN2	7	OUT2
	20	8	20	IN2_GND	8	OUT2_GND
	21	9	21	IN3	9	OUT3
	22	10	22	IN3_GND	10	OUT3_GND
	23	11	23	IN4	11	OUT4
	24	12	24	IN4_GND	12	OUT4_GND

#11 : Pin assignment for serial I/O ports

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



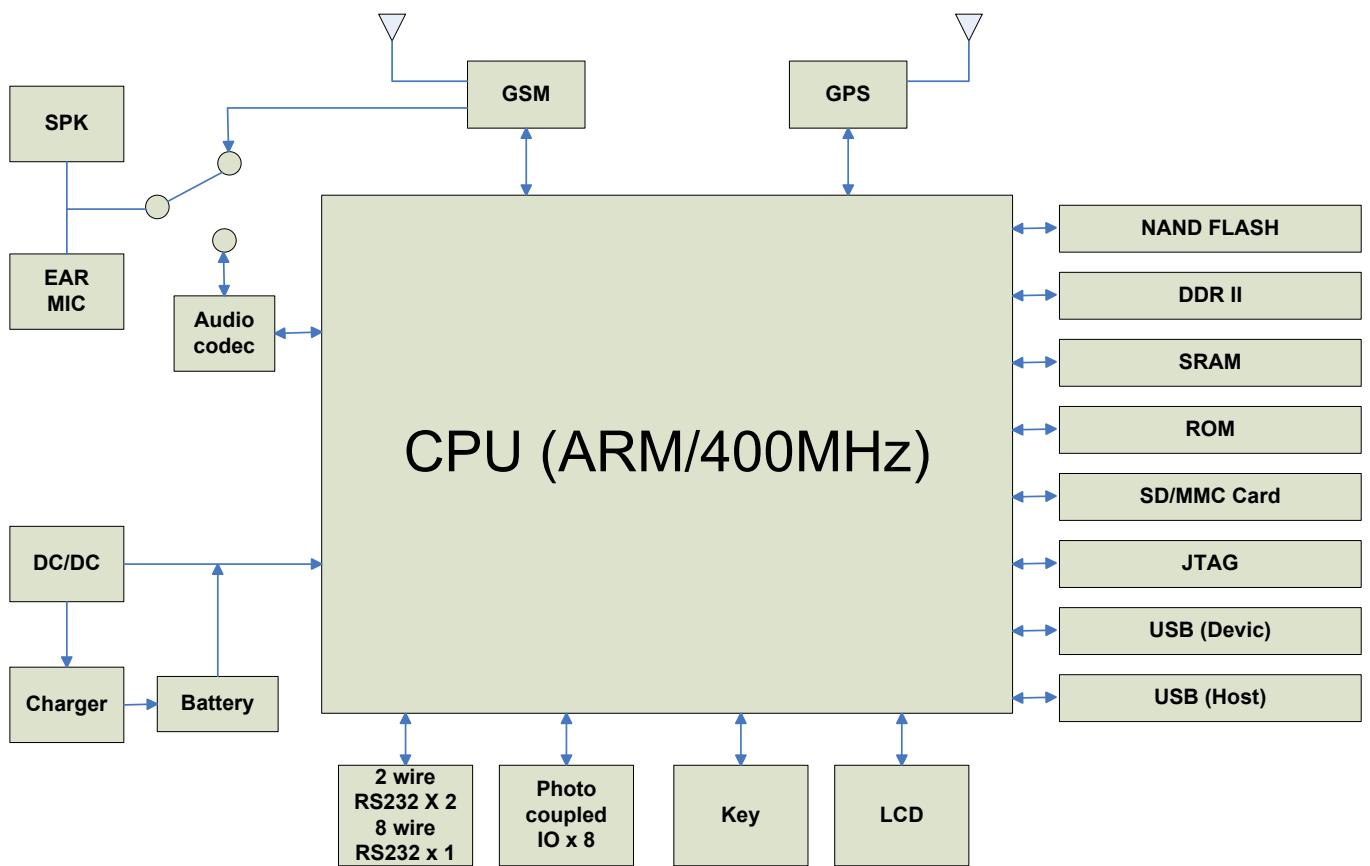
#	Name	Description
14	GSM Antenna	Connects to GSM
15	SD Card port	SD/MMC Card slot (up to 4GB max)



#	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	<ul style="list-style-type: none"><li>• Samsung S3C2416X 400MHz</li><li>• 32 bit RISC architecture ARM926EJ CPU core</li></ul>
System Memory	<ul style="list-style-type: none"><li>• 16-bit 64MB/133MHz DDR2 memory</li><li>• 512K Byte SRAM</li></ul>
Storage Device	2GB NAND Flash
Series Port	<ul style="list-style-type: none"><li>• 2-wire RS232 x 2</li><li>• 8-wire RS232 x1</li></ul>
USB	<ul style="list-style-type: none"><li>• 1x USB device (USB2.0)</li><li>• 1XUSB host (USB1.0)</li></ul>
GPIO	Supply 4 photo-coupler input and 4 photo-coupler output
Audio	<ul style="list-style-type: none"><li>• Support DSP based processing stereo codec with SNR 102 dB DAC/differential microphone input</li><li>• Dual channels 2 watts speaker output</li></ul>
Integrated Modules	<ul style="list-style-type: none"><li>• GPS Module (LEA-5S)</li><li>• GSM Module (SIEMENS MC-55)</li></ul>
Supply OS	WinCE 5.0(default)
LCD Size	7" TFT LCD
LCD Response	800 x 480 RGB
LCD Brightness	400 cd/m <sup>2</sup>
Power Supply	DC9V~DC28V
Operating Temperature	-30°C ~ +80°C

### 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	<table border="0"> <tr> <td>Sky View</td> <td>Open Sky<sup>2</sup></td> <td>Indoor<sup>3</sup></td> </tr> <tr> <td>Module</td> <td>ALL</td> <td>LEA-5H/LEA-5S LEA-5A</td> </tr> <tr> <td>Cold Start(Autonomous)</td> <td>29 s</td> <td></td> </tr> <tr> <td>Hot Start(Autonomous)</td> <td>&lt;1 s</td> <td>10 s TBD</td> </tr> <tr> <td>Aided Start</td> <td>&lt;1 s</td> <td>10 s 10 s</td> </tr> <tr> <td>Reacquisition</td> <td>&lt;1 s</td> <td>10 s TBD</td> </tr> </table>	Sky View	Open Sky <sup>2</sup>	Indoor <sup>3</sup>	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
Sky View	Open Sky <sup>2</sup>	Indoor <sup>3</sup>																	
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	<table border="0"> <tr> <td>Tracking &amp; Navigation</td> <td>-160 dBm</td> </tr> <tr> <td>Acquisition &amp; Reacquisition</td> <td>-160 dBm</td> </tr> <tr> <td>Cold Start(Autonomous)</td> <td>-145 dBm</td> </tr> </table>	Tracking & Navigation	-160 dBm	Acquisition & Reacquisition	-160 dBm	Cold Start(Autonomous)	-145 dBm												
Tracking & Navigation	-160 dBm																		
Acquisition & Reacquisition	-160 dBm																		
Cold Start(Autonomous)	-145 dBm																		
Horizontal Position Accuracy <sup>4</sup>	<table border="0"> <tr> <td>Autonomous</td> <td>&lt;2.5 m</td> </tr> <tr> <td>SBAS</td> <td>&lt;2.0 m</td> </tr> </table>	Autonomous	<2.5 m	SBAS	<2.0 m														
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)																		

<sup>2</sup> All satellites at -130dB

<sup>3</sup> All satellites at -155dBm

<sup>4</sup> 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/°C
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

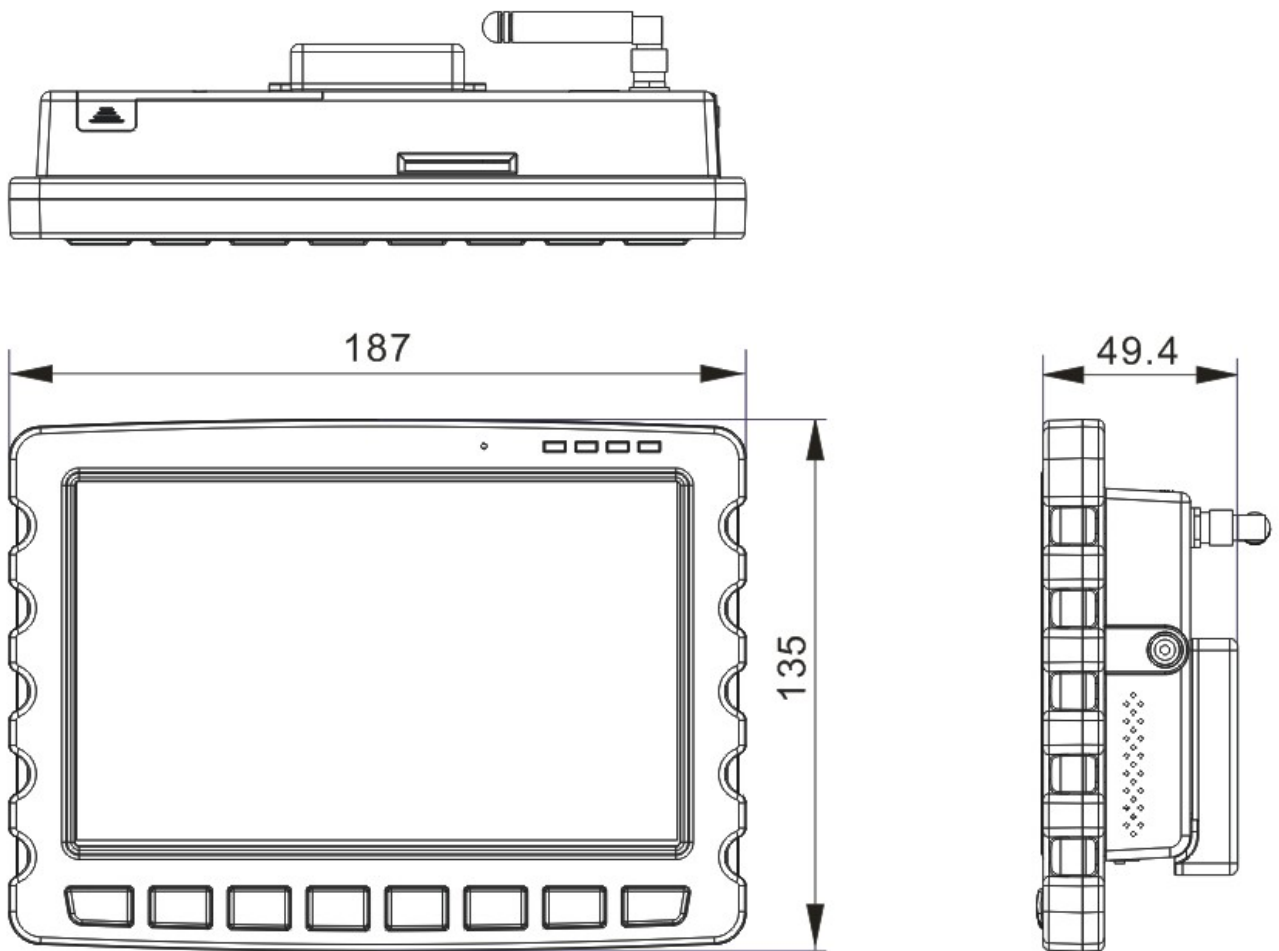
Parameter	Specifications
Frequency band	<ul style="list-style-type: none"> <li>MC55 Tri-band: EGSM 900, GSM 1800, GSM 1900</li> <li>Compliant to GSM Phase 2/2+</li> </ul>
GSM class	Small MS
Transmit power	<ul style="list-style-type: none"> <li>Class 4 (2W) at EGSM 900 and GSM 850</li> <li>Class 1 (1W) at GSM 1800 and GSM 1900</li> </ul>
GPRS connectivity	<ul style="list-style-type: none"> <li>GPRS multi-slot class 10</li> <li>GPRS mobile station class B</li> </ul>
Ambient operating temperature according to IEC 60068-2	<ul style="list-style-type: none"> <li>Normal operation: -20°C to +55°C</li> <li>Restricted operation: -25°C to -20°C and +55°C to +70°C</li> <li>Automatic thermal shutdown: <math>\leq -25^\circ\text{C}</math> and <math>\geq +70^\circ\text{C}</math></li> </ul> <p>When an emergency call is in progress automatic temperature shutdown is deferred</p>
Humidity	Max. 90% relative humidity
GPRS data	<ul style="list-style-type: none"> <li>GPRS data downlink transfer: max. 85.6 kbps</li> <li>GPRS data uplink transfer: max. 42.8 kbps</li> <li>Coding scheme: CS1, CS2, CS3 and CS4</li> <li>MC55/MC56 Support the two protocols PAP (Password Authentication Protocol) and CHAP (Challenge Handshake Authentication Protocol) commonly used for PPP connections.</li> <li>Support of Packet Switched Broadcast Control Channel (PBCCH) allows you to benefit from enhanced GPRS performance when offered by the network operators.</li> </ul>
CSD data	<ul style="list-style-type: none"> <li>CSD transmission rates: 2.4, 4.8, 9.6, 14.4 kbps, non-transparent, V.110</li> <li>Unstructured Supplementary Services Data (USSD) support</li> </ul>
SMS	<ul style="list-style-type: none"> <li>MT, MO, CB, Text and PDU mode</li> <li>SMS storage: SIM card plus 25 SMS locations in the mobile equipment</li> <li>Transmission of SMS alternatively over CSD or GPRS. Preferred mode can be user-defined</li> </ul>
TCP/IP stack	Internet services: TCP, UDP, HTTP, FTP, SMTP, POP3 Access by AT commands
FAX	Group 3: Class 1, Class 2
SIM interface	<ul style="list-style-type: none"> <li>Supported SIM card: 3V</li> <li>External SIM card reader has to be connected via interface connector</li> </ul>
External antenna	Connected via 50 Ohm antenna connector or antenna pad
Audio interfaces	Two analog audio interfaces, one digital audio interface(DAI)
Audio features	<p>Speech codec modes:</p> <ul style="list-style-type: none"> <li>Half Rate (ETS 06.20)</li> <li>Full Rate(ETS 06.10)</li> <li>Enhanced Full Rate(ETS 06.50/06.60/06.80)</li> <li>Adaptive Multi Rate(AMR)</li> </ul> <p>Hands free operation</p> <ul style="list-style-type: none"> <li>Echo cancellation</li> <li>Noise reduction</li> </ul>

## 1.2.6. GSM antenna specifications

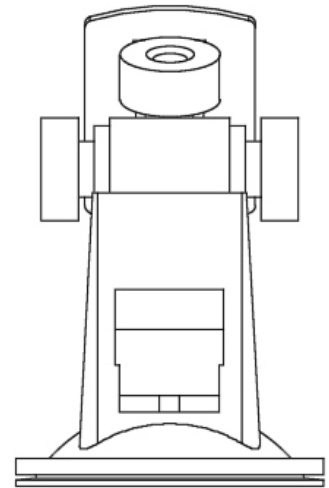
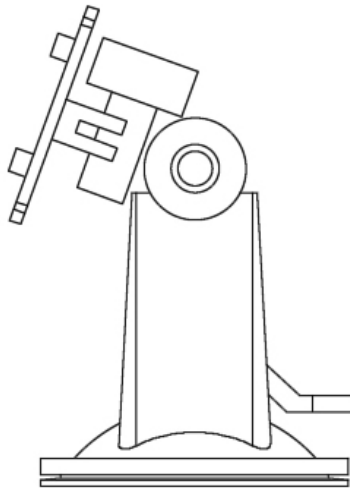
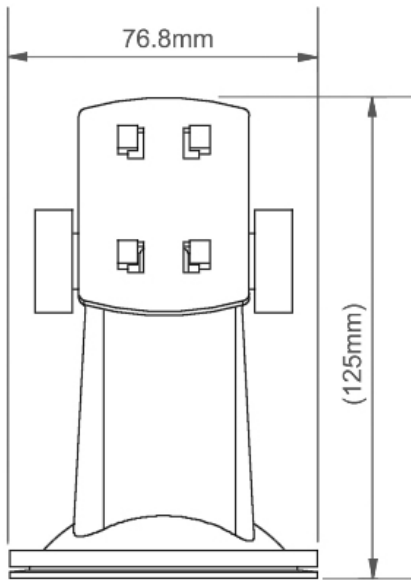
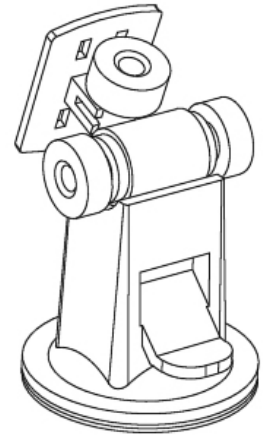
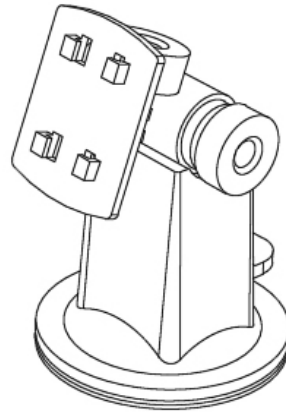
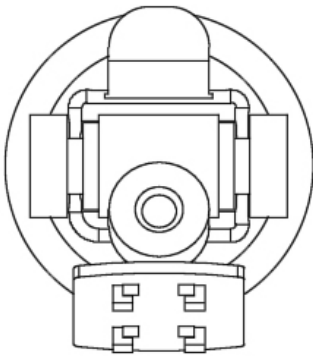
Parameter	Specifications
Frequency Range	880~960 MHz and 1710~1990 MHz
Impedance	50 ohm
VSWR	$\leq 3.5$
Gain	0 dBi (Max)
Polarization	Vertical
Radiation pattern	Near omni-directional in the horizontal plane

## 1.3. Mechanical specifications

### 1.3.1. Mechanical specifications of module



### 1.3.2. Mechanical specifications of Holder & Mount



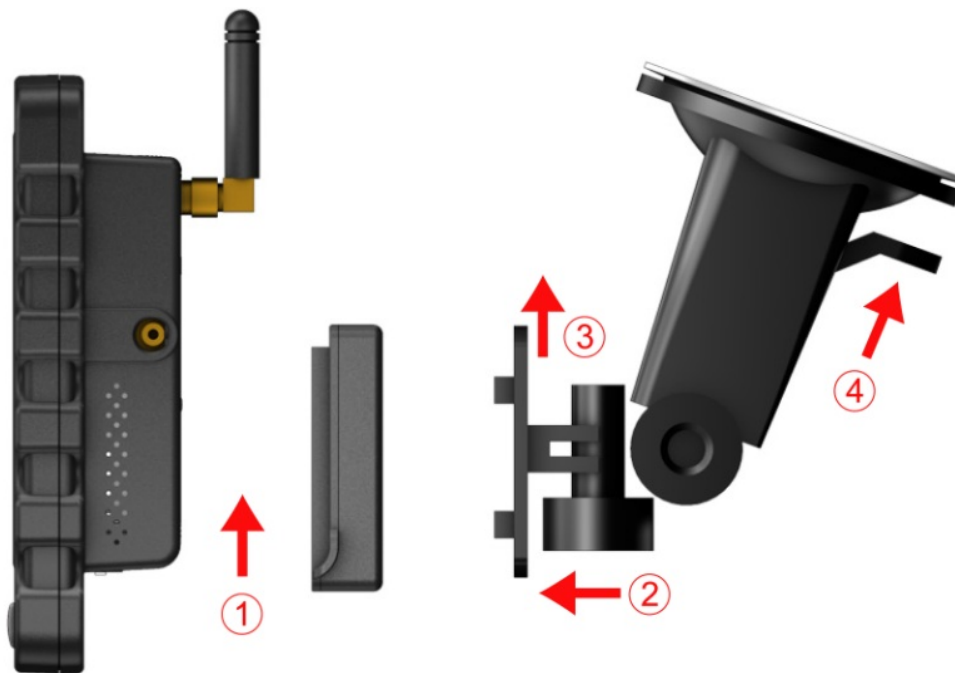
## 2. MCP-903A Installation guide

The chapter provides guidance for hardware installation.

- 2.1 Install sucker and mount
- 2.2 Install GSM Antenna
- 2.3 Install SIM Card
- 2.4 Install SD Card
- 2.5 Install external GPS antenna
- 2.6 Install battery
- 2.7 Connect power
- 2.8 MCP-903A power status

## 2.1. Install a sucker & mount

Follow the sequence shown below, install MCP-903A 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.



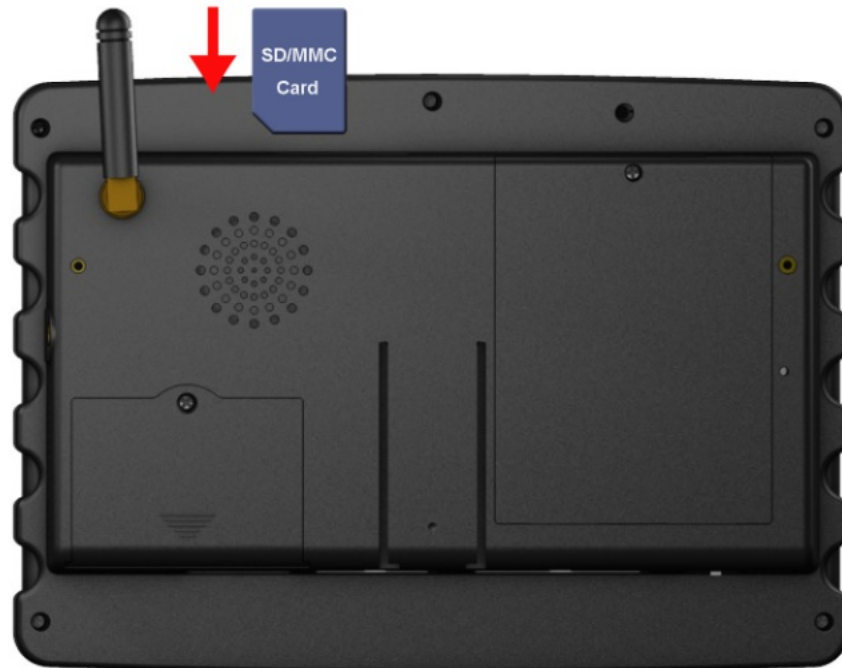
## 2.3. Install a SIM card

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

MCP-903A 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 MCP-903A 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.

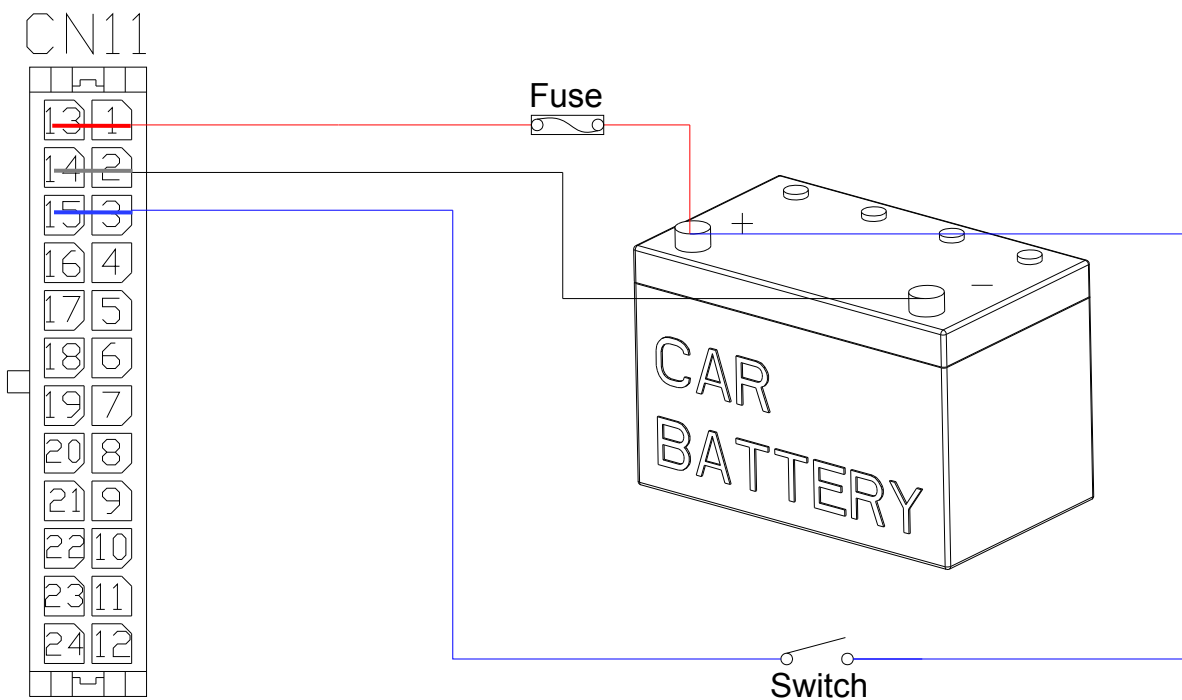
Note that only a 3.7V/2700mA Lithium battery is applicable, and typically with 300 to 350 recharge cycles.



## 2.7. Connect Power

The operating voltage of MCP-903A 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 on MCP-903A to boot up.



Note:

A quick troubleshooting hint – If MCP-903A 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 MCP-903A module for repair.

## 2.8. MCP-903A power status

When no (optional) battery is installed in MCP-903A, the power is support by car battery – that is, car switch needs to be “on” position and power switch of BETA903A need to be on to boot up. If optional battery is applicable, simply power on MCP-903A 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	MCP-903A Power switch	MCP-903A Battery Status	RTC Battery Status	MCP-903A 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 MCP-903A are both OFF, the RTC battery inside MCP-903A 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 MCP-903A off, MCP-903A battery and RTC battery will get charged from car battery.

**Mode 3-** Turn car switch on, then turn MCP-903A is on. Same as mode 2, MCP-903A battery and RTC battery will get charged from car battery.

**Mode 4-** Turn car switch off, Same as mode 1, MCP-903A 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 MCP-903A

This chapter provides a step guide to test I/O on MCP-903A module and it is broken down into:

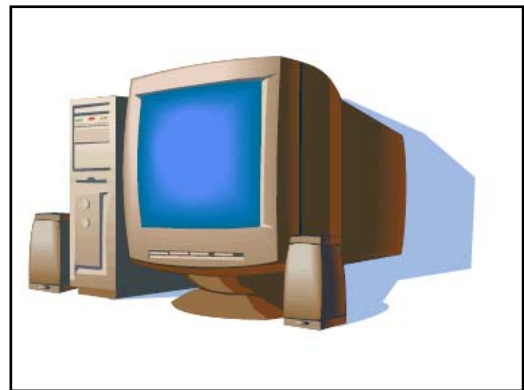
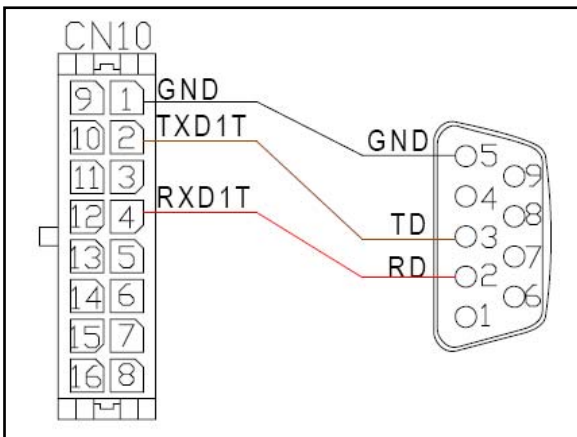
- 3.1 Serial port test
- 3.2 GSM test
- 3.3 GPS test
- 3.4 GPIO test
- 3.5 Keypad test

# 3.1. Serial Port test

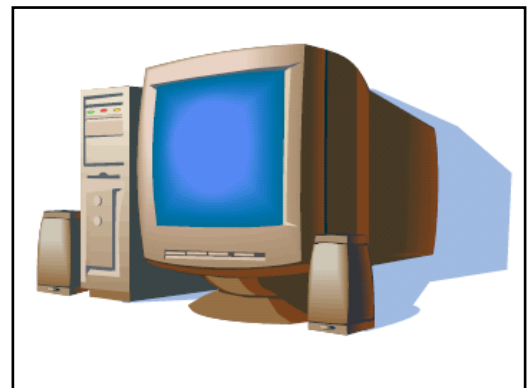
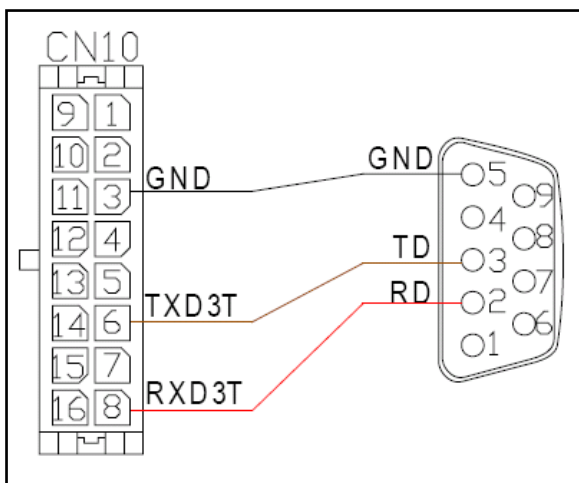
## 3.1.1. Connect Serial Port to PC

MCP-903A supports two sets of 2-wire RS232 and one set of 8-wire RS232 for serial interface to a PC.

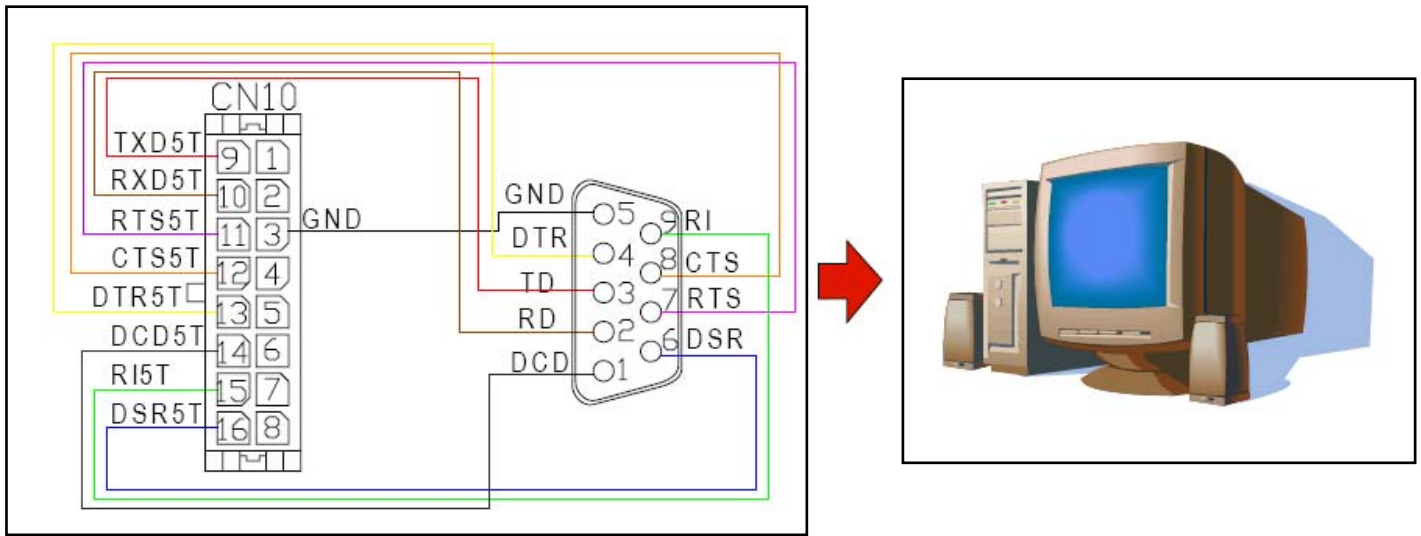
**1<sup>st</sup> set of 2-wire RS232 (Debug Port)** - Via a RS-232 serial cable, connect the 1<sup>st</sup> set to COM port of PC. This port is defaulted to a debug port and not for external use. Its signal level is at +/- 12 volt.



**2<sup>nd</sup> set of 2-wire RS232 (COM3)** - Via a RS-232 serial cable, connect the 2<sup>nd</sup> 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.



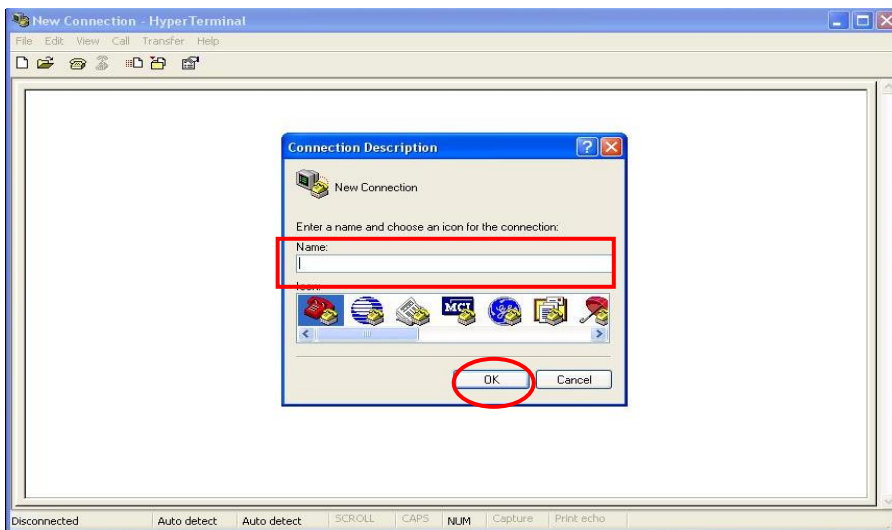
The following table shows the pin assignment mapping for all available serial ports.

Table of RS232 Pin Assignment							
RS232 DB9 Connector		1 <sup>st</sup> 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

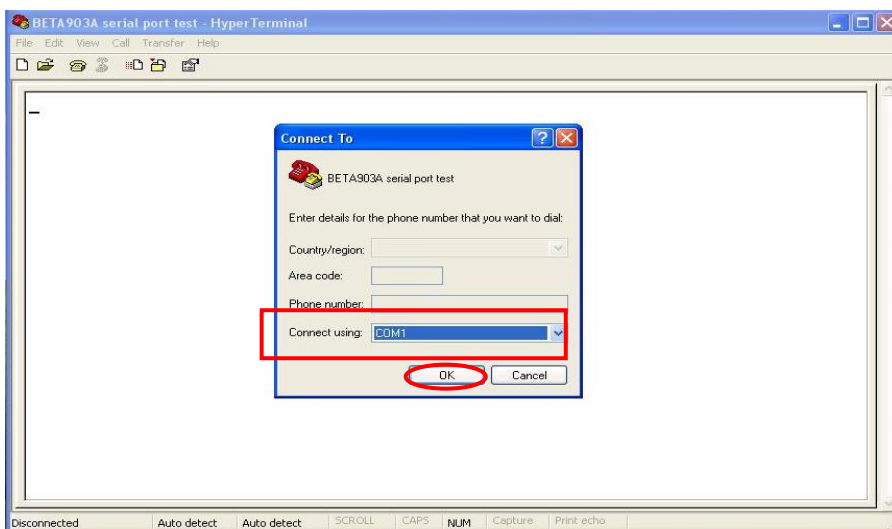
### 3.1.2. Serial port test

Upon completion of serial connection of MCP-903A 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.

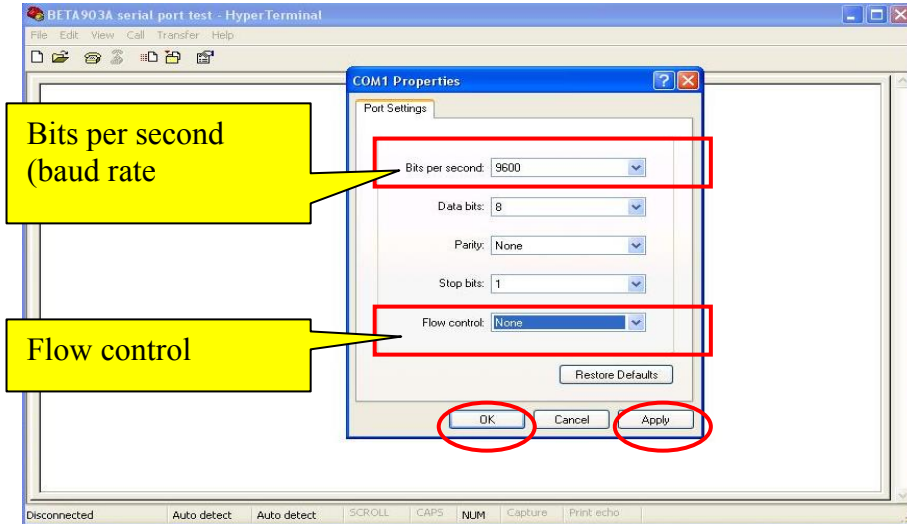
**STEP1:** Run the hyper-terminal on PC, then enter MCP-903A, click on OK.



**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 MCP-903A. A typical baud rate of MCP-903A ranges from 9600 to 115200 and use “None” for flow control. Click on Apply, then OK.

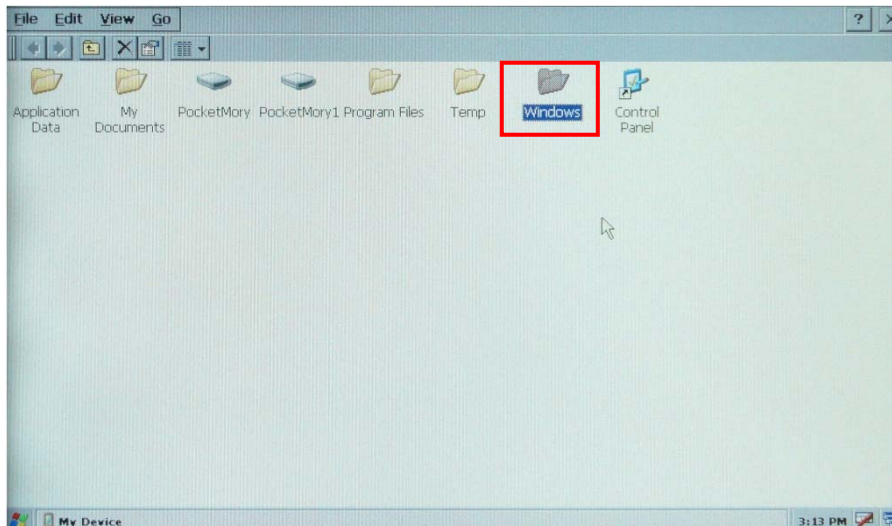


## MCP-903A set up:

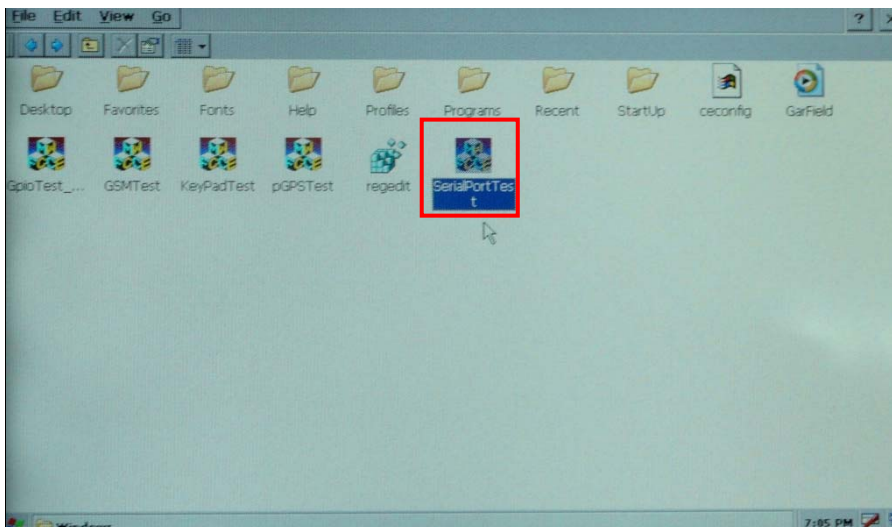
**STEP1:** After booting up MCP-903A, on Windows CE desktop, click on "My Device" ◦



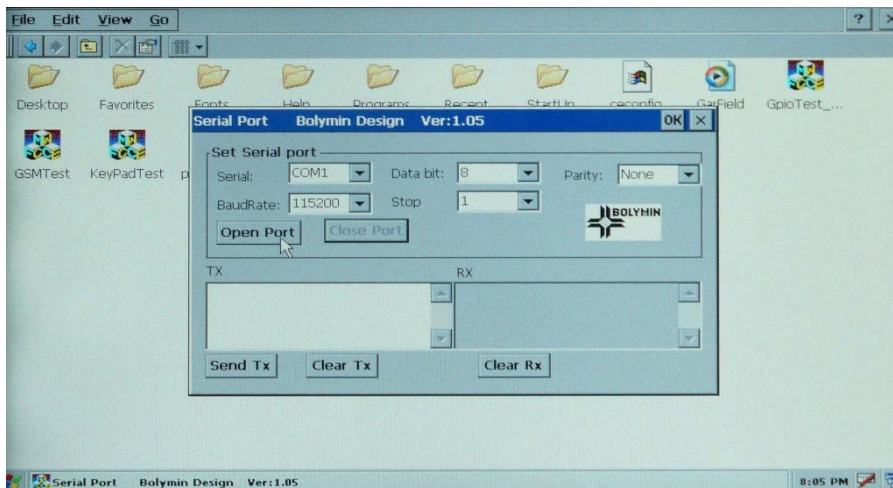
**STEP2:** Select "Windows" folder



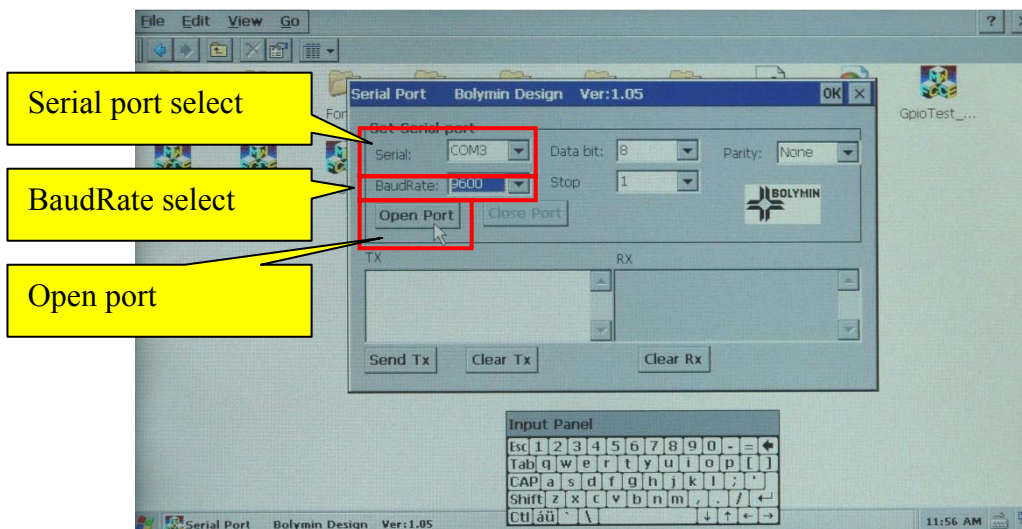
**STEP3:** Run "SerialPortTest" program



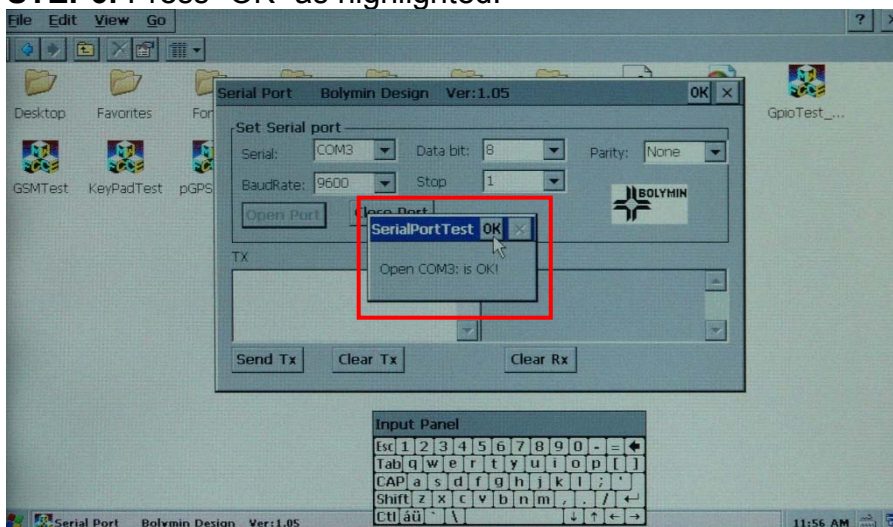
**STEP4:** A "SerialPortTest" running screen.



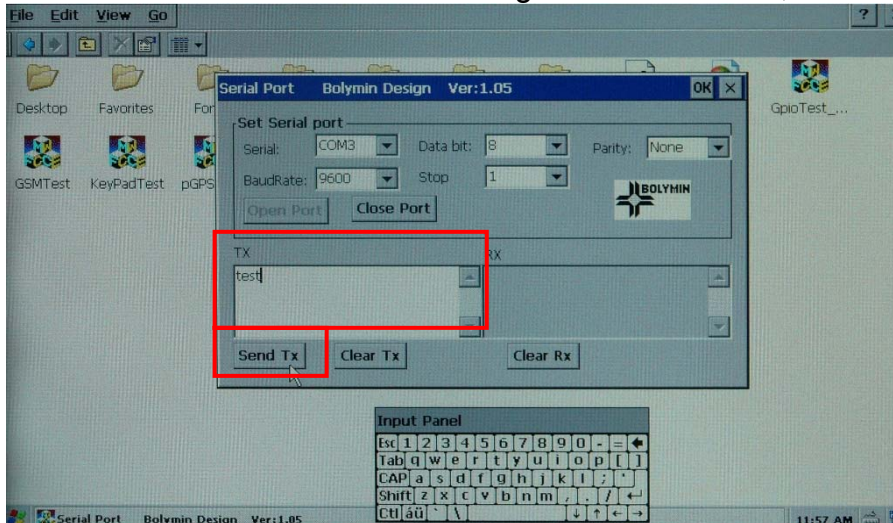
**STEP5:** Please make sure the baud rate setting on PC and MCP-903A are identical with a typical range of 9600 to 115200. then clock on "Open Port" to connect to PC.



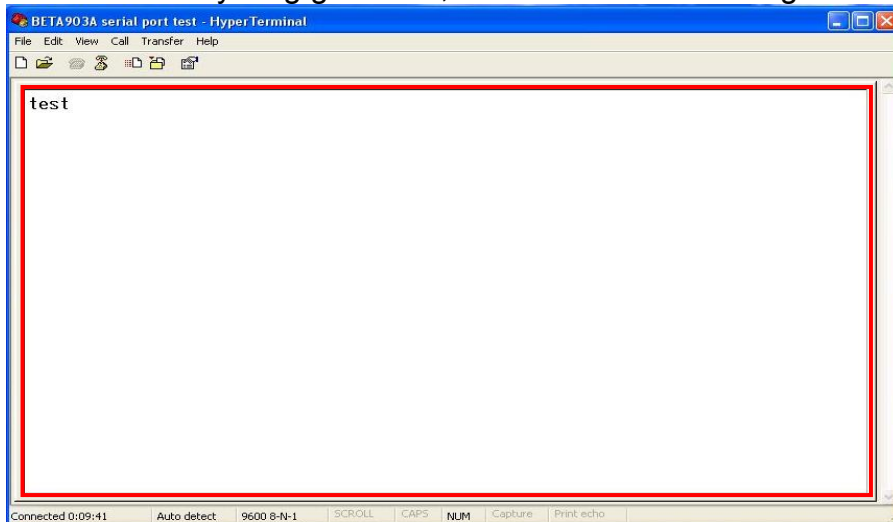
**STEP6:** Press "OK" as highlighted.



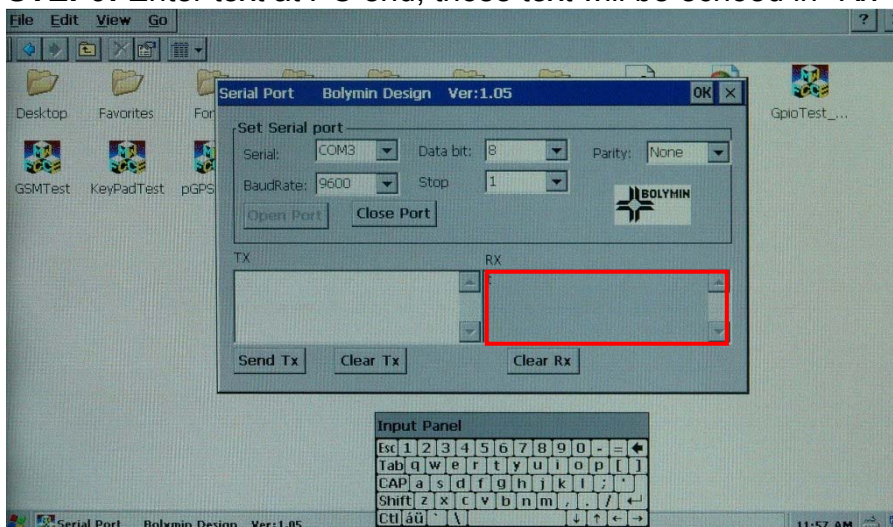
**STEP7:** Enter some trial text message on the “TX” box, then click on “Send Tx” .



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



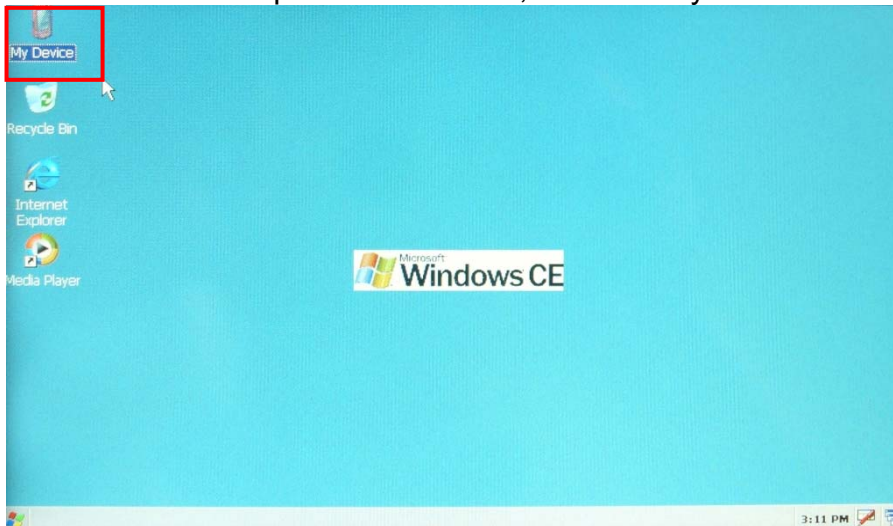
**STEP9:** Enter text at PC end, those text will be echoed in “Rx” box at MCP-903A.



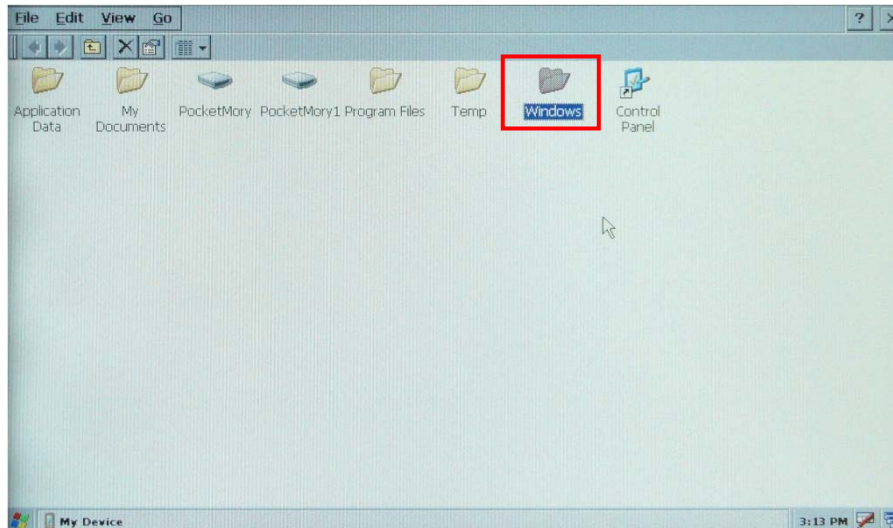
## 3.2. GSM test

### Test procedure:

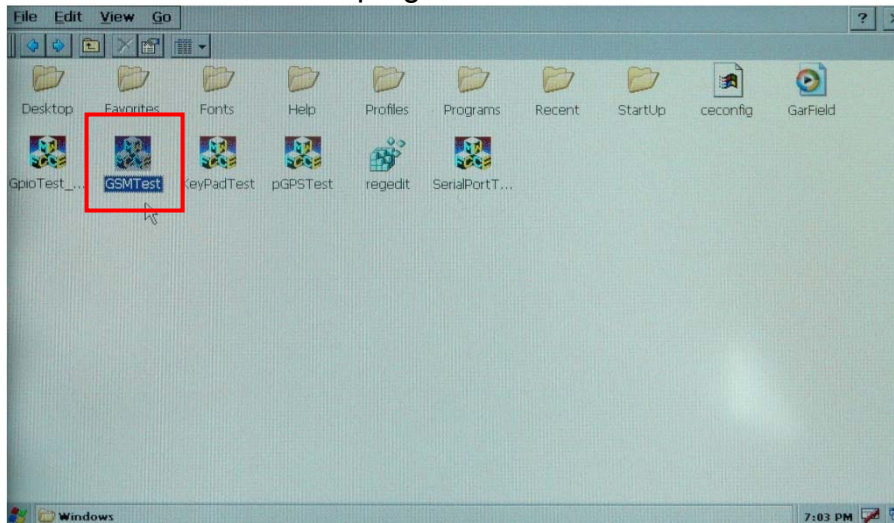
**STEP1:** On desktop of Windows CE, click on "My Device"



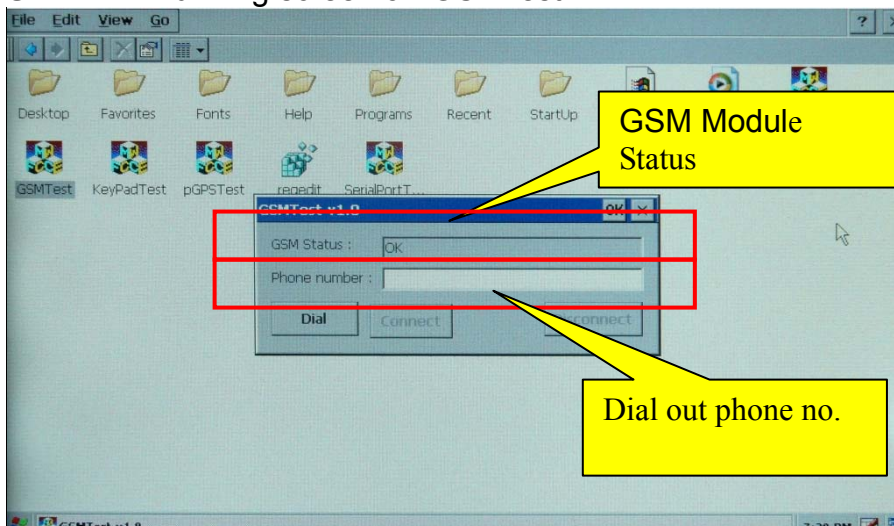
**STEP2:** Select "Windows" folder



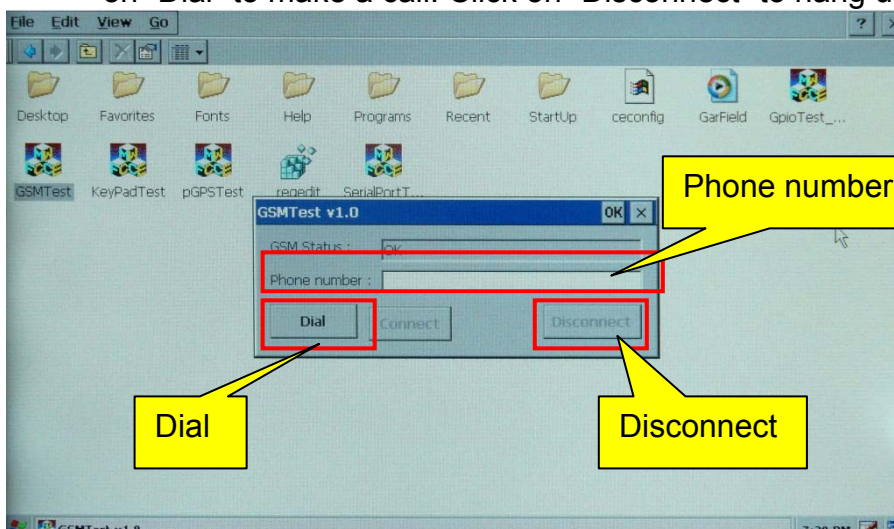
**STEP3:** Run "GSMTTest" program.



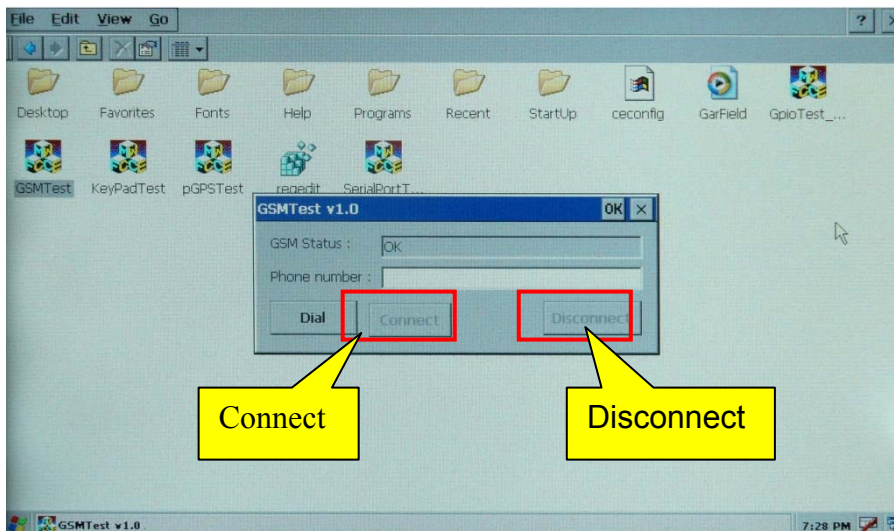
**STEP4:** Running screen of "GSMTTest"



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



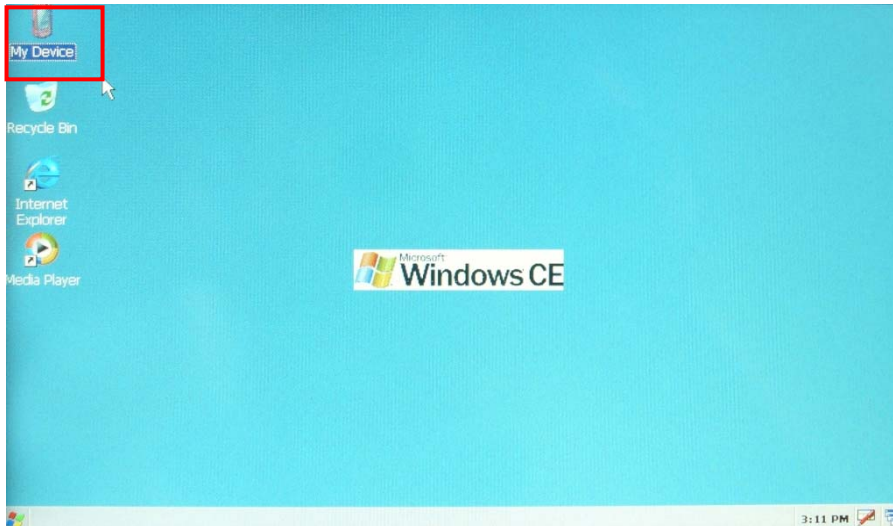
**STEP6:** MCP-903A will pop up a screen show a "RING" at status bar when an incoming call occurs. Click on "Connect" to pick up the phone. Click on "Disconnect" to hang up.



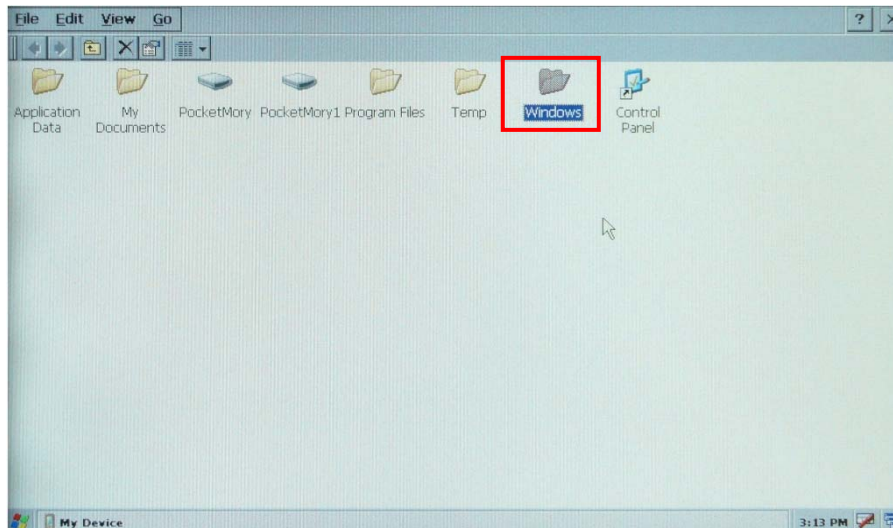
## 3.3. GPS test

### Test Procedure:

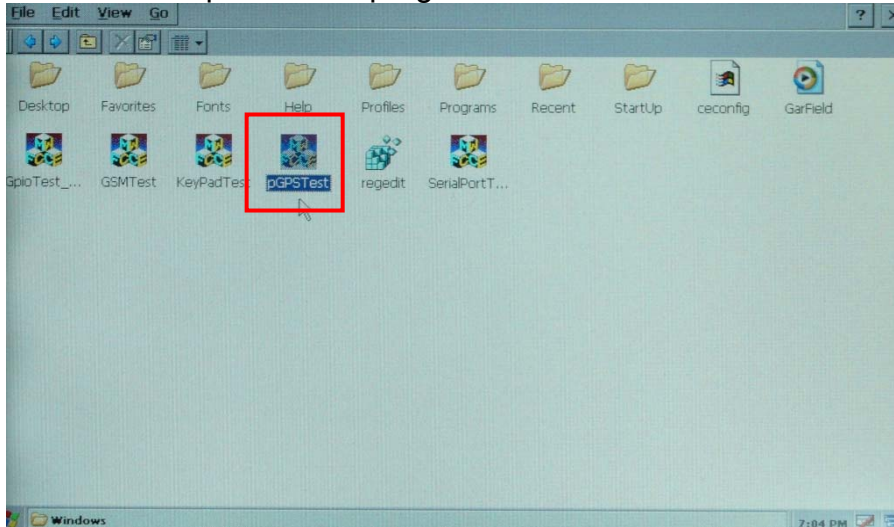
**STEP1:** On desktop of Window CE, click on “My Device” ◦



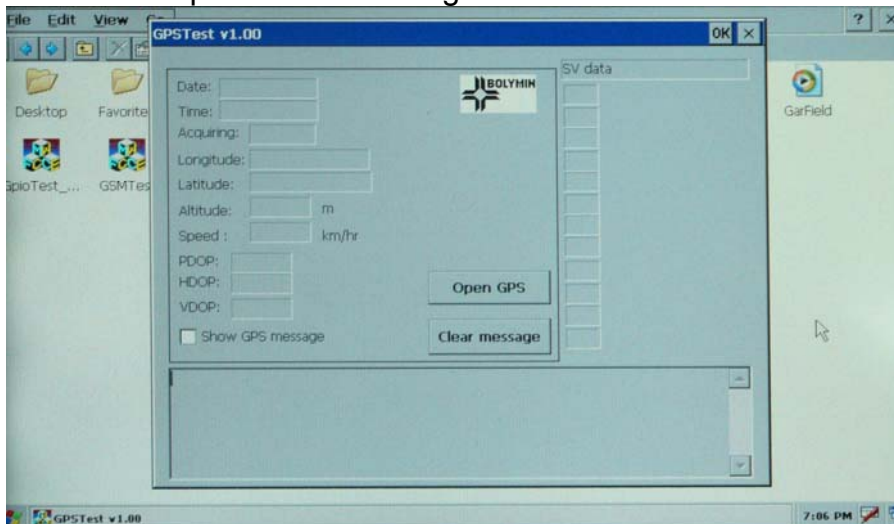
**STEP2:** Click on “Windows” folder



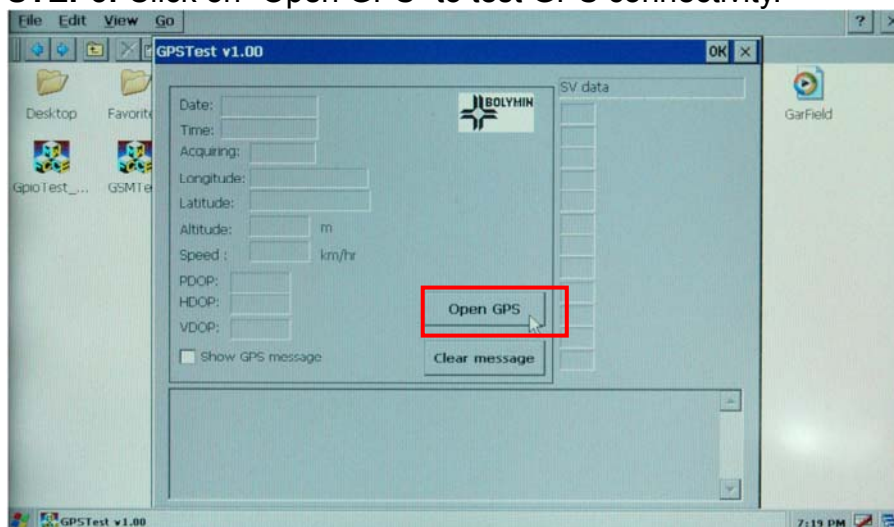
**STEP3: Run "pGPSTest" program**



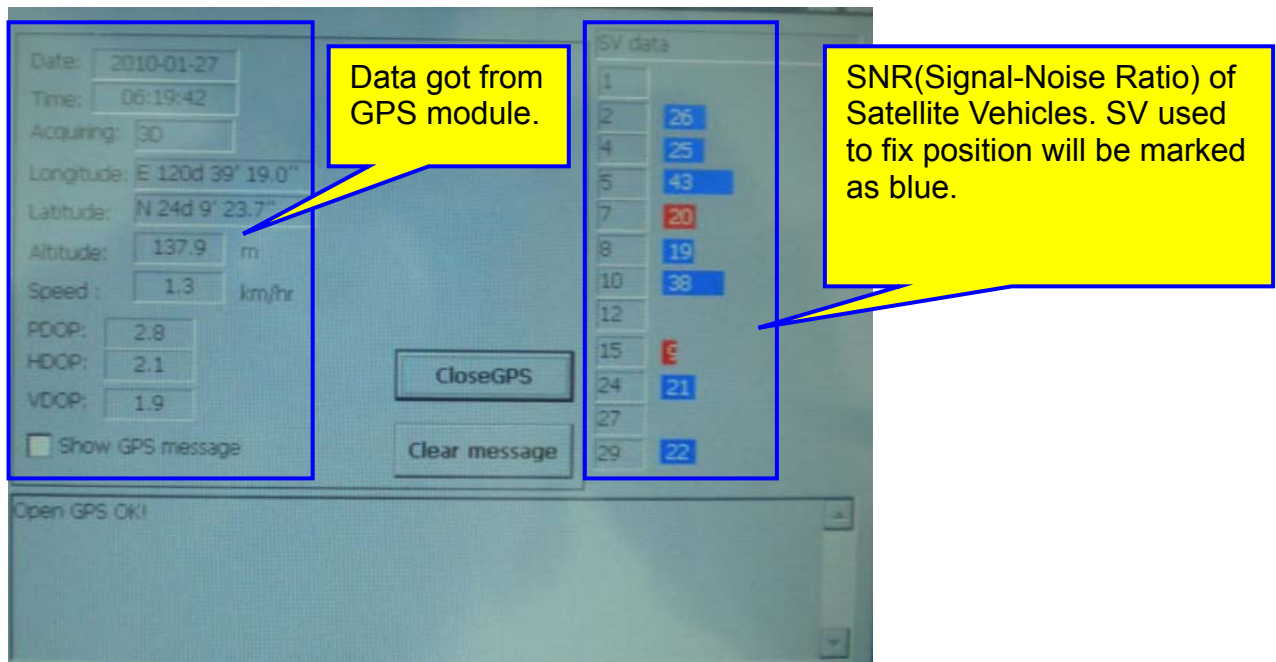
**STEP4: A "pGPSTest" running screen**



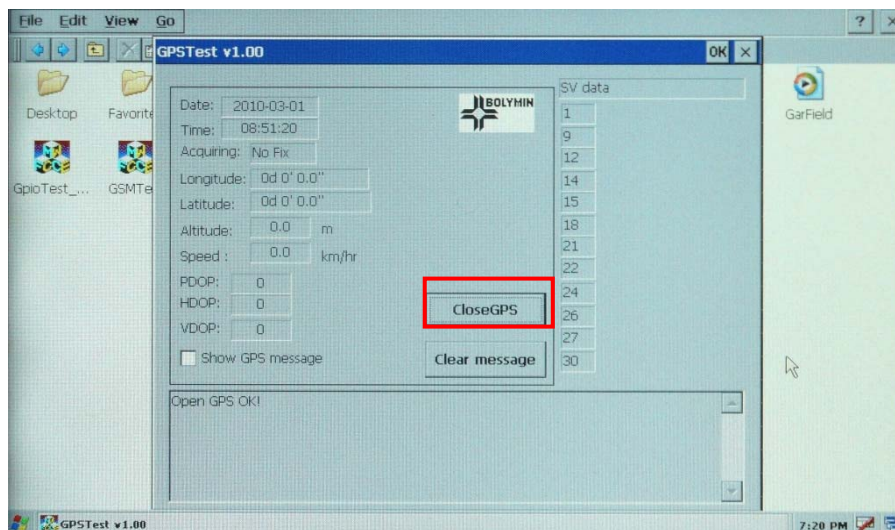
**STEP5: Click on "Open GPS" to test GPS connectivity.**



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



**STEP7:** Click on "Close GPS" to terminate GPS connection

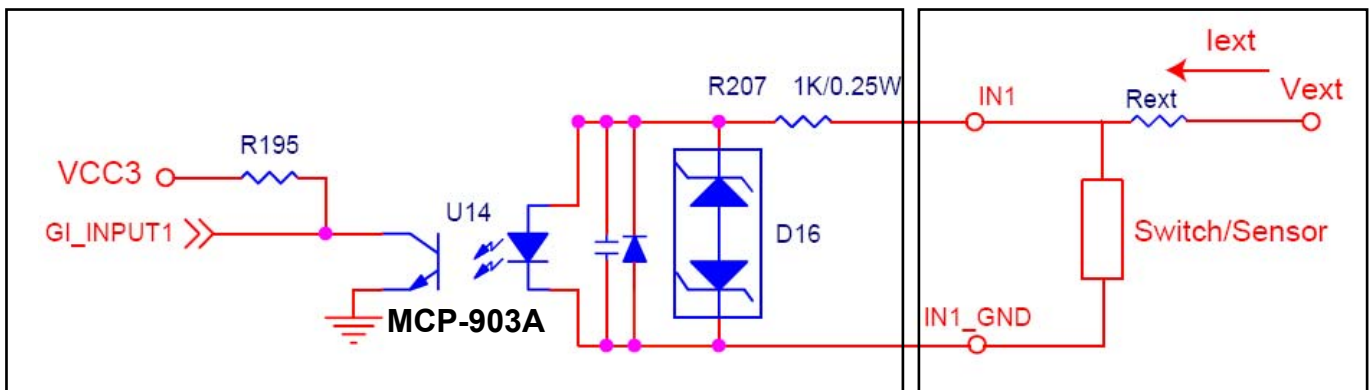


## 3.4. GPIO test

### 3.4.1. Recommended interfacing circuit to GPIO

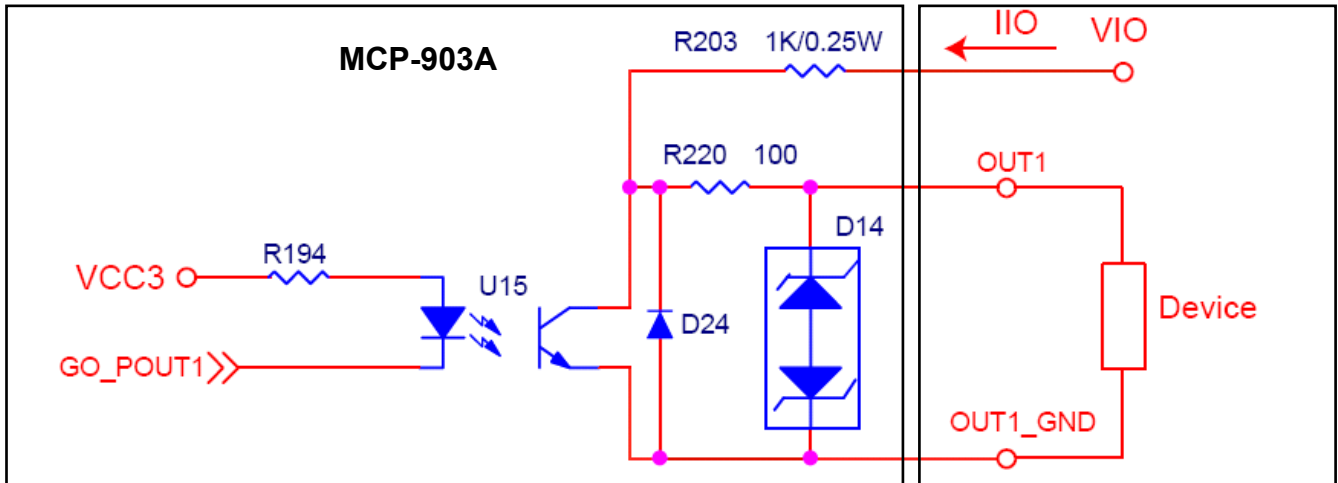
The MCP903A supports 4 sets of photo-coupler inputs and 4 sets of photo-coupler outputs for general purpose inputs/outputs. Those I/Os are useful to read out signals and access control devices.

#### Input circuit examples:



Note: The value of  $R_{ext}$  depends on that of  $V_{ext}$ . The rule of thumb is keep  $I_{ext}(\text{min current})$  at a typical value of 5mA, and  $I_{ext}(\text{max})$  at 14mA. If  $R_{ext}=0$ , set  $V_{ext}$  to 12 volt, then  $I_{ext}$  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.



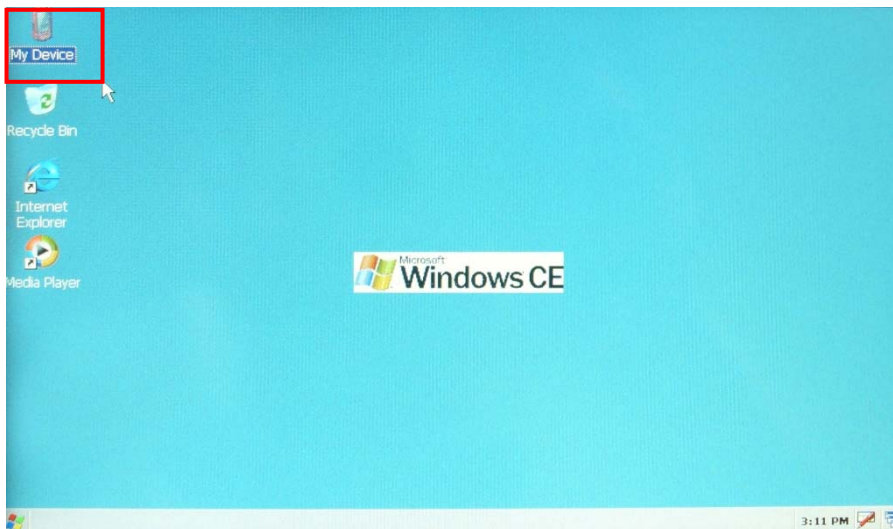
Note: The voltage and current value of OUT1 pin depends upon VIO and Rex, if any. Use caution to keep IIO(min current) to 5mA and IIO(max) to 14mA as rated.

## 3.4.2. GPIO test

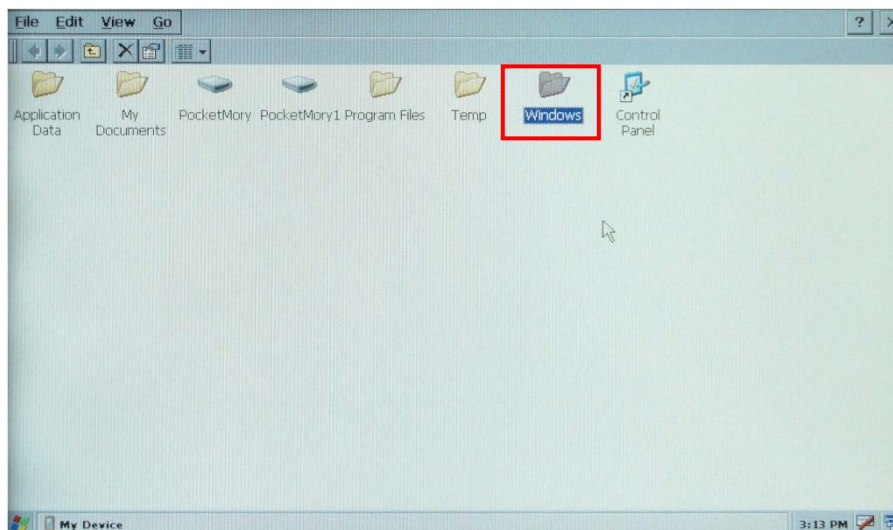
At the end of the chapter, we provide a build-in GPIO test program. Simply follows the recommended GPIO circuit as illustrated on Chapter 3.4.1, then start using this test program to validate the GPIO ports. ◦

### Test Procedure:

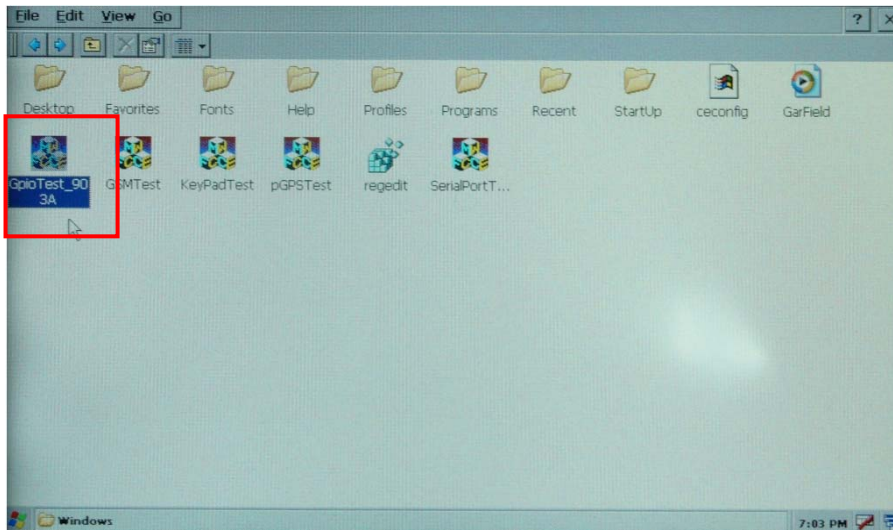
**STEP1:** On desktop of Windows CE, click on "My Device"



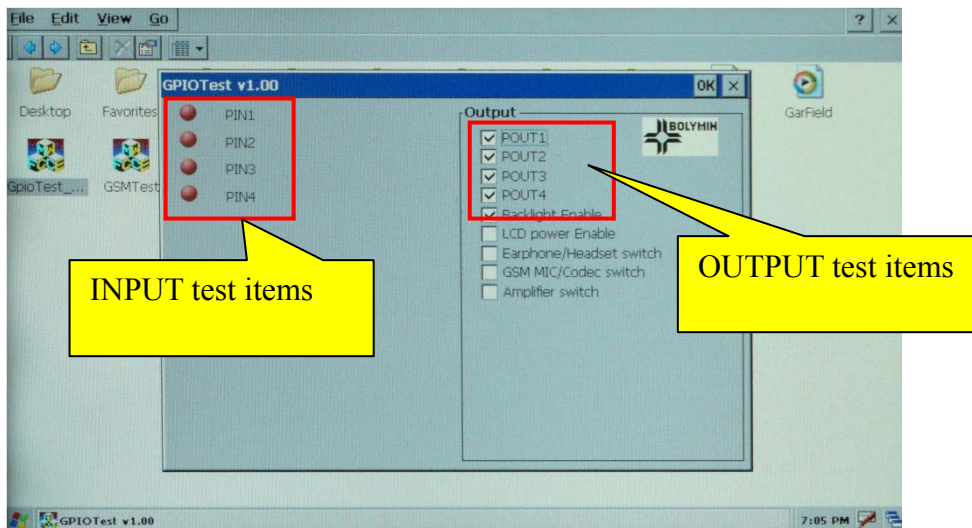
**STEP2:** select "Windows" folder



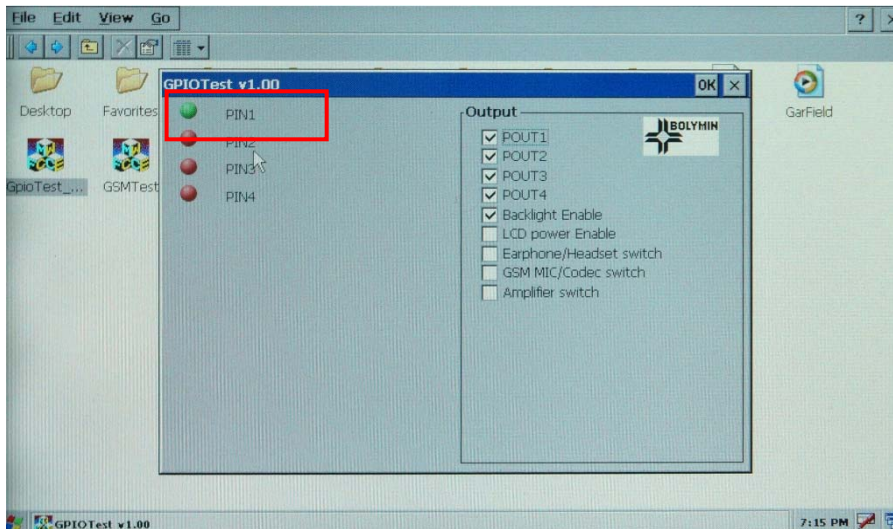
**STEP3:** Run "GPIO Test\_903A" program



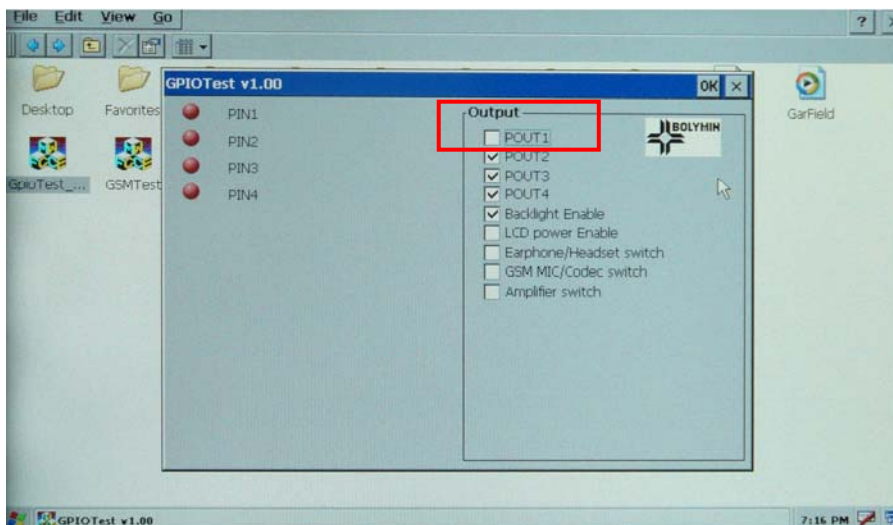
**STEP4:** A "GPIO Test\_903A" running screen



**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.



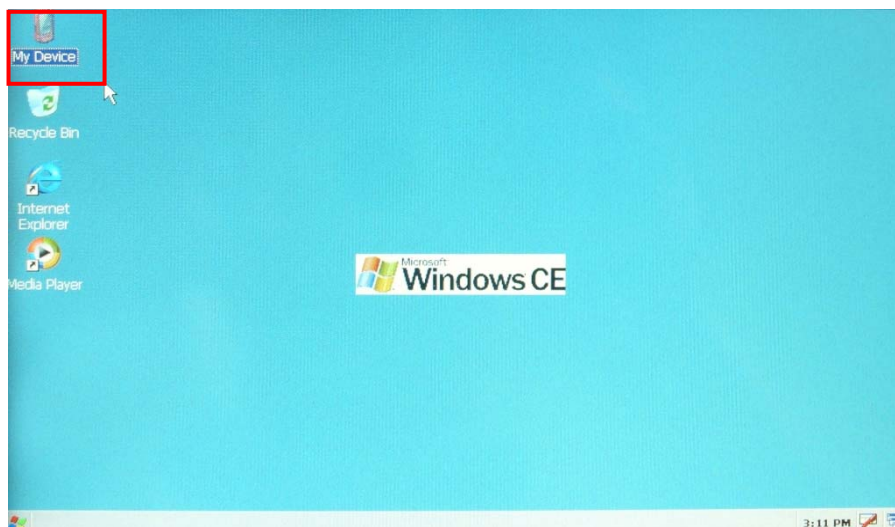
## 3.5. Keypad Test

AT ÖÜË€œ supports 8 custom-defined keypads, usu. defined as menu buttons.

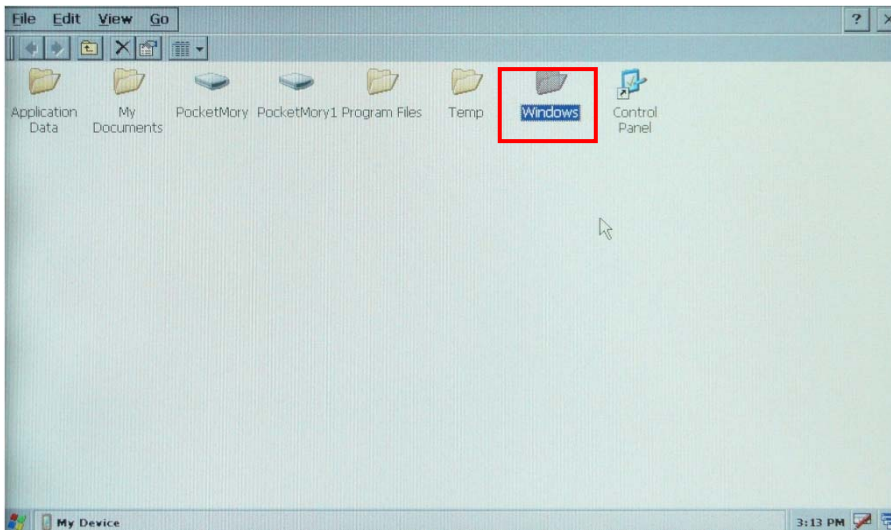


### Test Procedure:

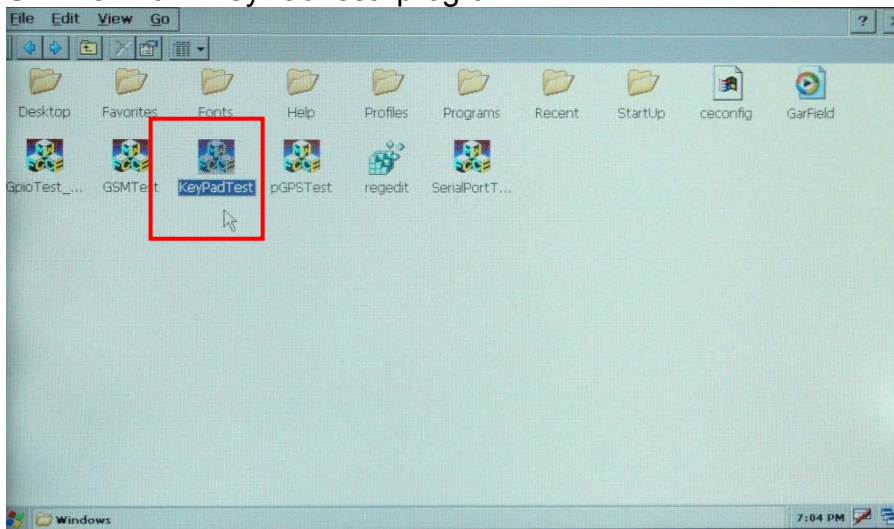
**STEP1:** On desktop of Windows CE, click on "My Device"



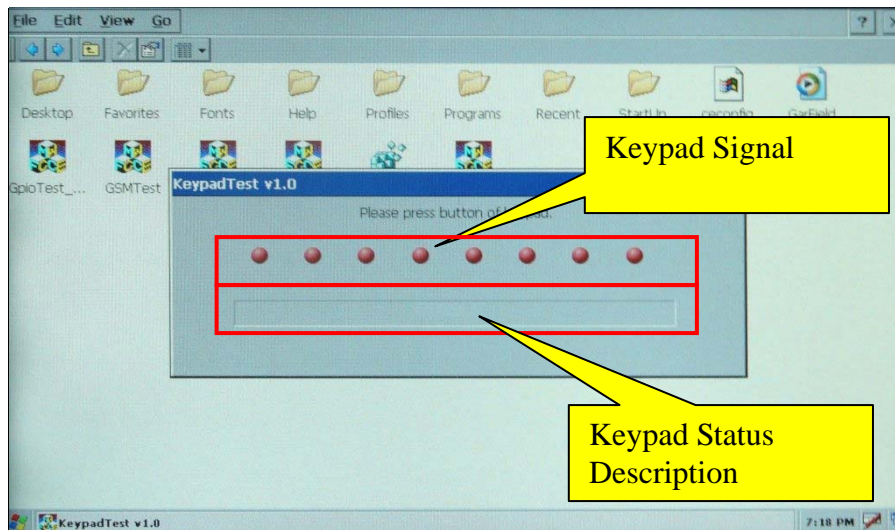
**STEP2:** Select "Windows" folder



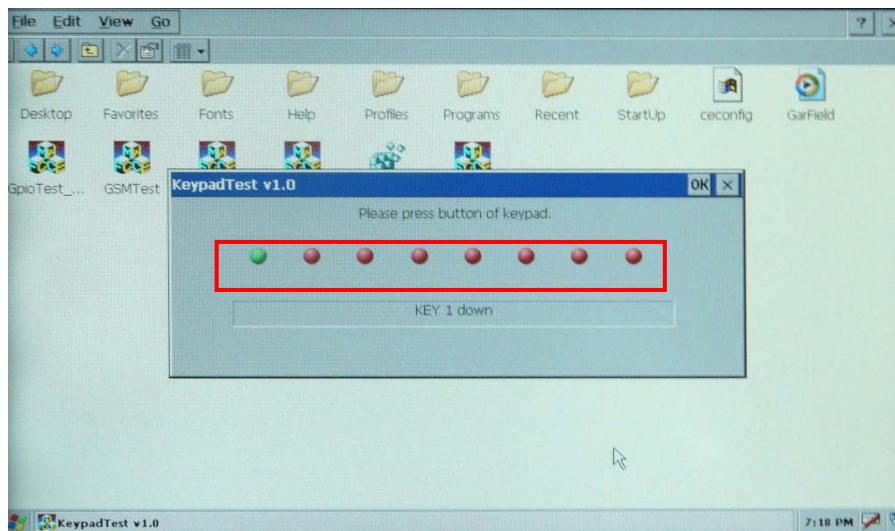
**STEP3:** Run "KeyPadTest" program



**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. MCP-903A 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 Transfer File Between MCP-903A And PC
- 4.2 Programming For MCP-903A
- 4.3 Serial Port Function
- 4.4 GSM Control
- 4.5 GSM Message
- 4.6 GPIO and Keypad Control

# 4.1. File Between MCP-903A And PC

## 4.1.1. Connect PC and MCP-903A

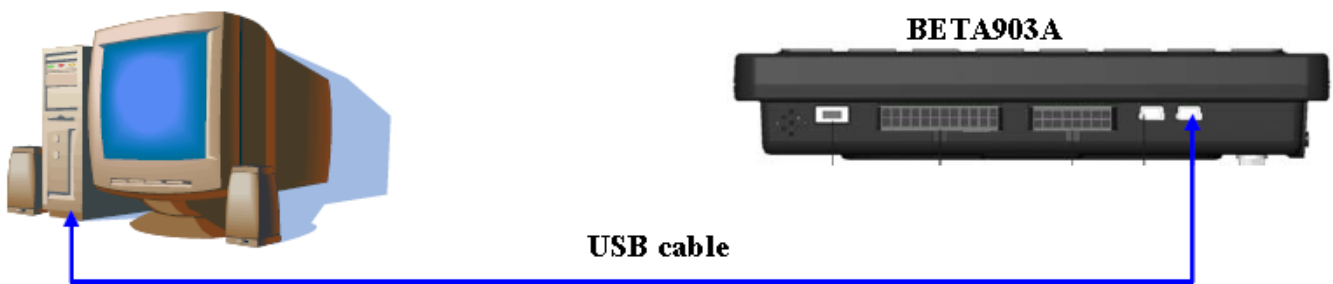
Here is a simple 3 steps to establish a connection between desktop PC and MCP-903A :

**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 MCP-903A by USB cable. The following picture shows the hardware connection between desktop PC and MCP-903A for file transfer.

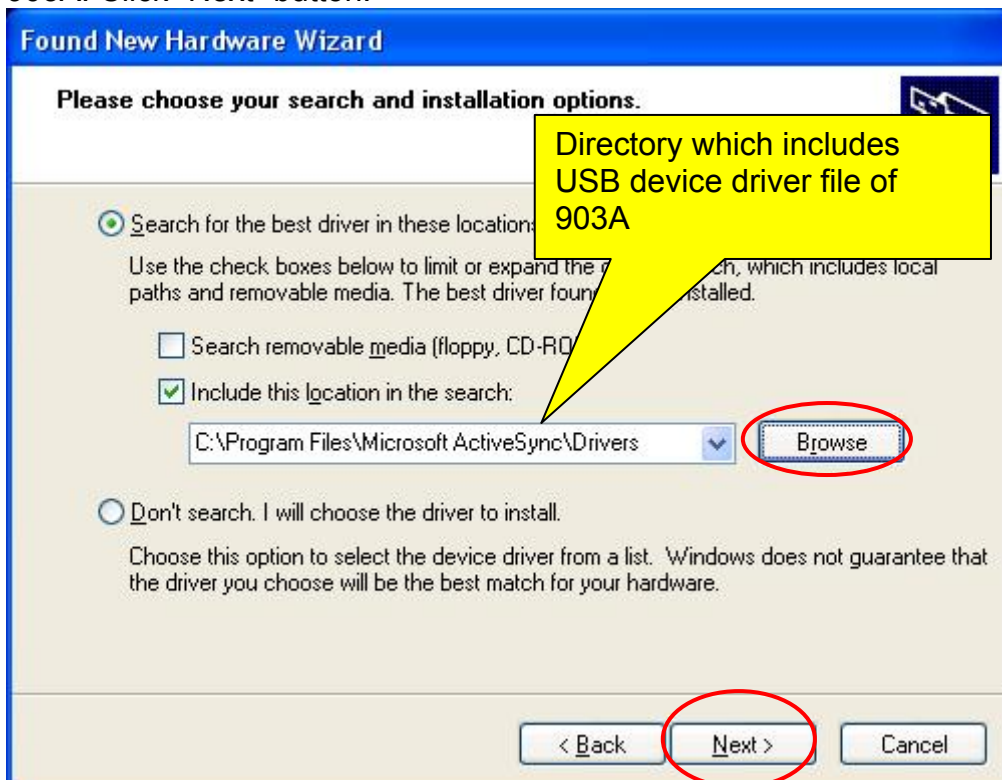


**STEP 3.** Connect T ÔÙË903A to PC through a USB cable, then power on T ÔÙË903A. For the first-time connection, windows system on PC will request for the USB device driver of T ÔÙË 903A. Please install USB driver as follows:

(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 T ÔÙË 903A. Click “Next” button.



(3). Click "Continue" button



(4). Click "Finish" button. Now T ÔÙÈ03A will connect to PC by ActiveSync.

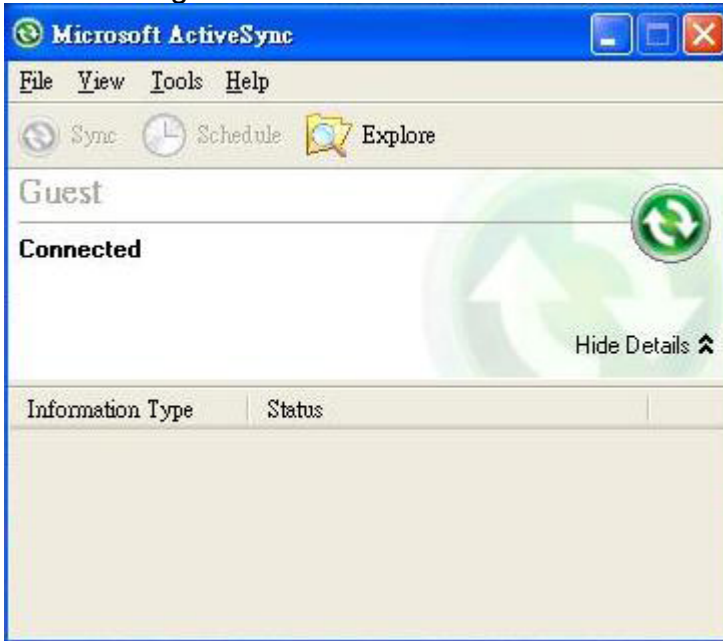


(5). Select “No” and click “Next” button to cancel the synchronization.

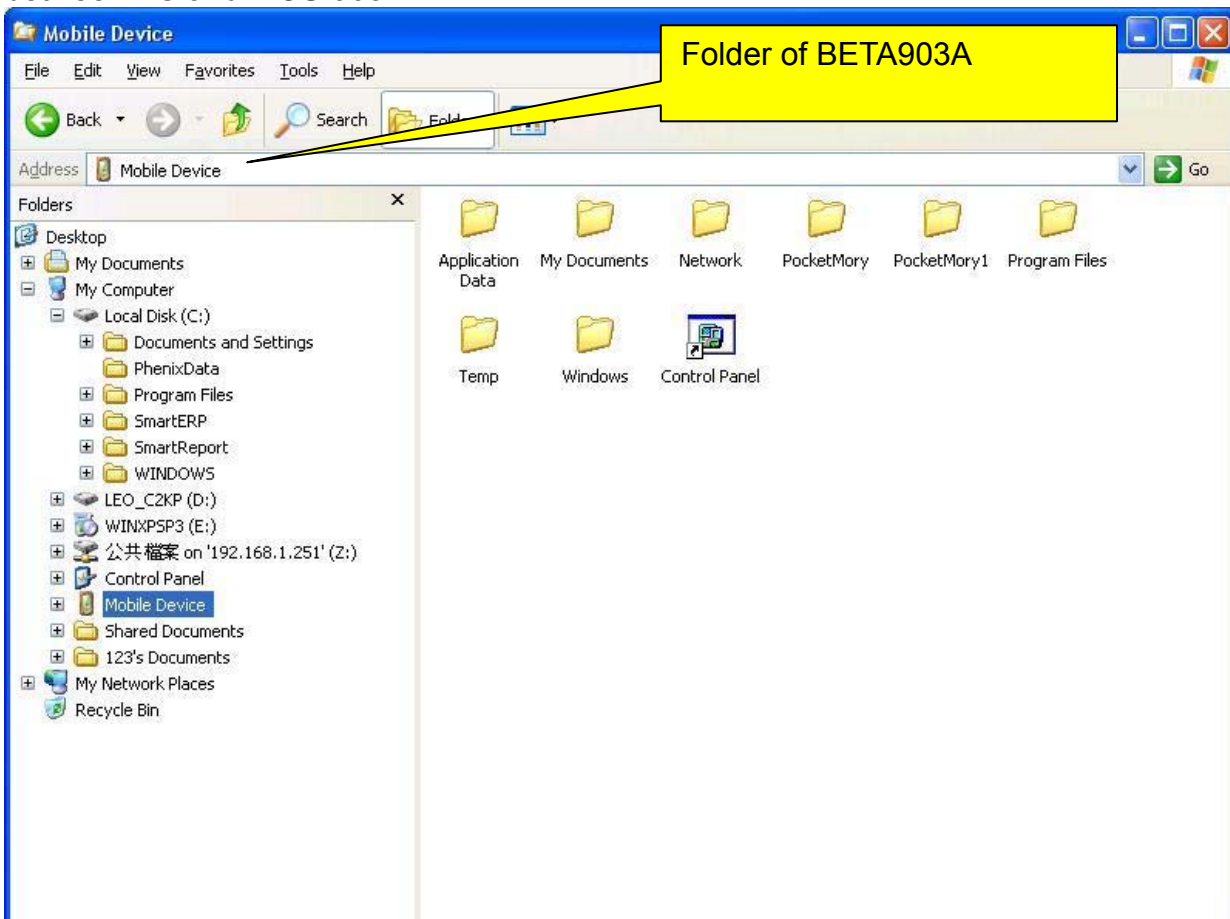


## 4.1.2. Transfer File

Once the USB connection is established, your ActiveSync application will show a “connected” status. The green circle means the connection between PC and T ÔÚË903A is done sucessfully.



Run "Explore" program and browse into the folder of T ÔÚË903A. You can easily transfer files between PC and T ÔÚË903A



## 4.2. Programming For A7 D!903A

### 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

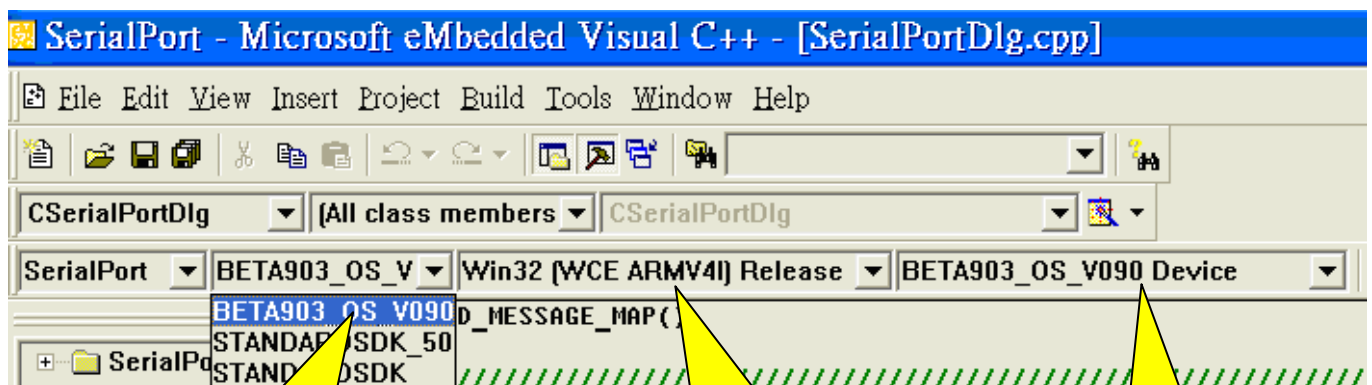
<http://www.microsoft.com/downloads/details.aspx?FamilyID=1DACDB3D-50D1-41B2-A107-FA75AE960856&displayLang=en.>

2. Connect A7 D!903A and Desktop PC by procedures in section 4.1.

3. Install SDK of A7 D!903A provided by Bolymin. 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::



Select T ÔÚÉ903A platform

Debug or release version

T ÔÚÉ903A device

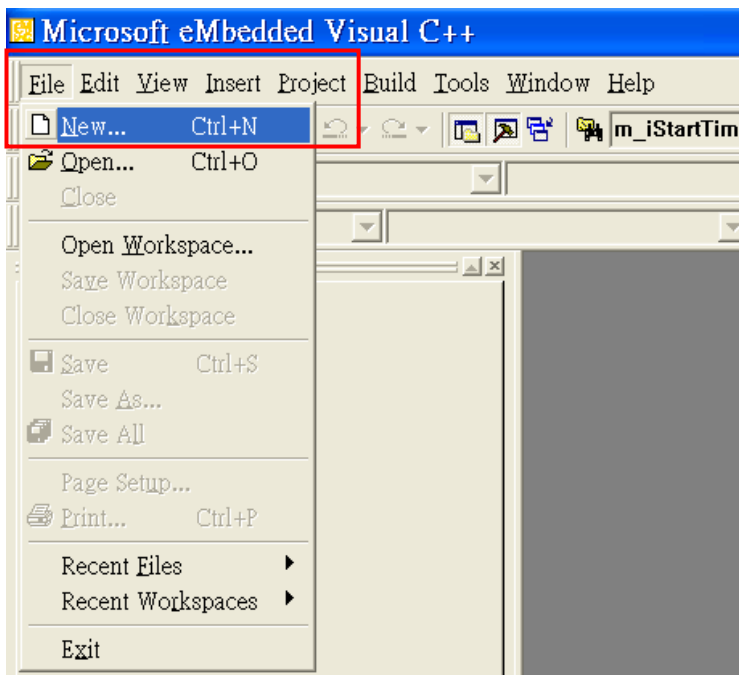
## 4.2.2. Create New Project

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.

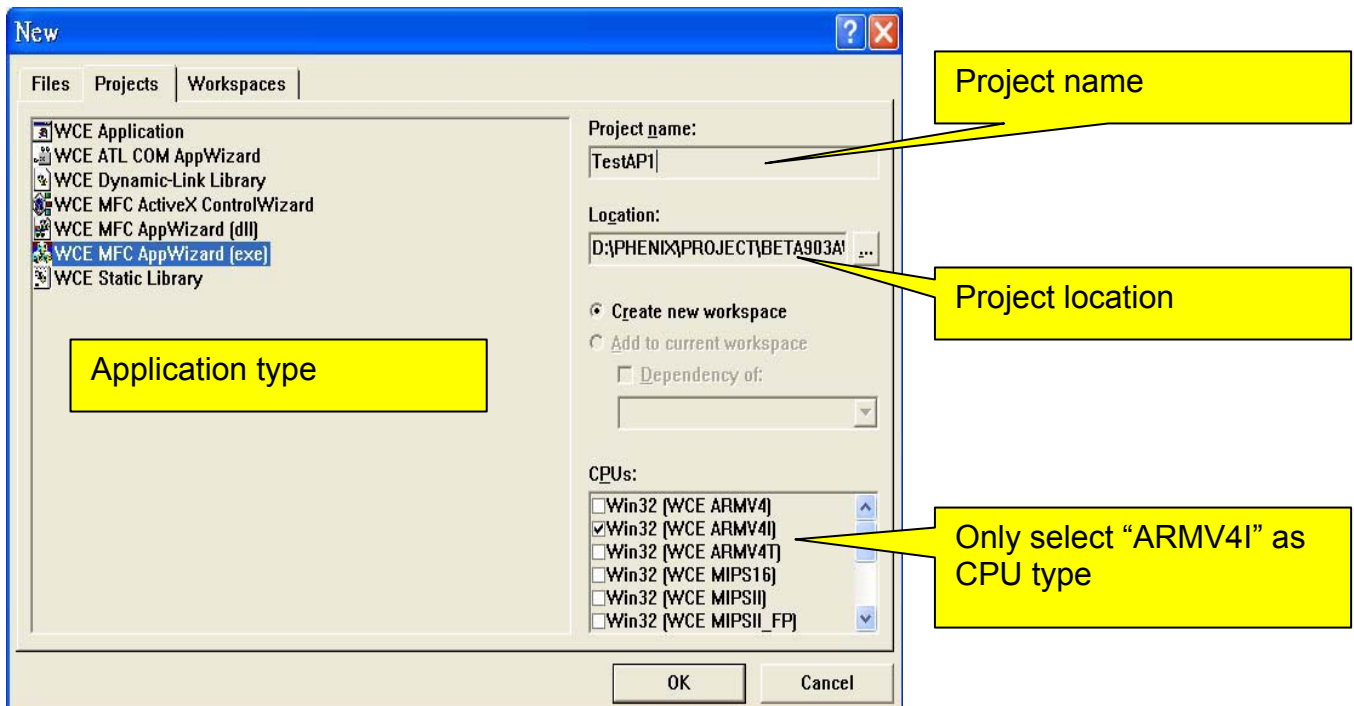
You could create a new project for your application as follows:

**STEP 1:** Execute eVC 4.0.

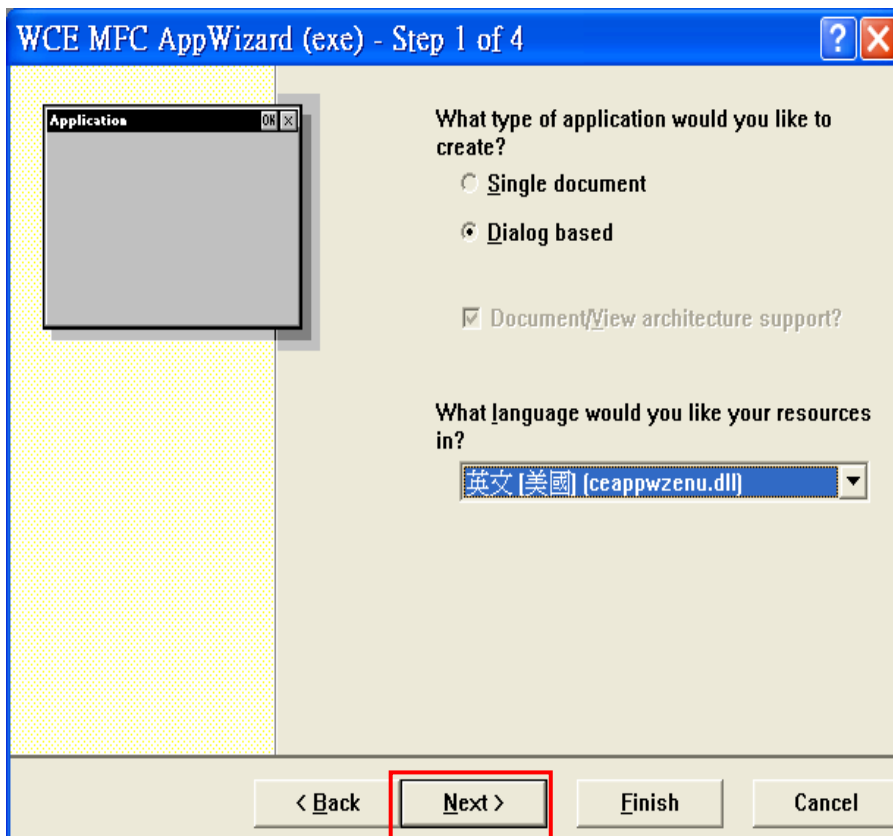
**STEP 2:** Select “File”-“New...: function



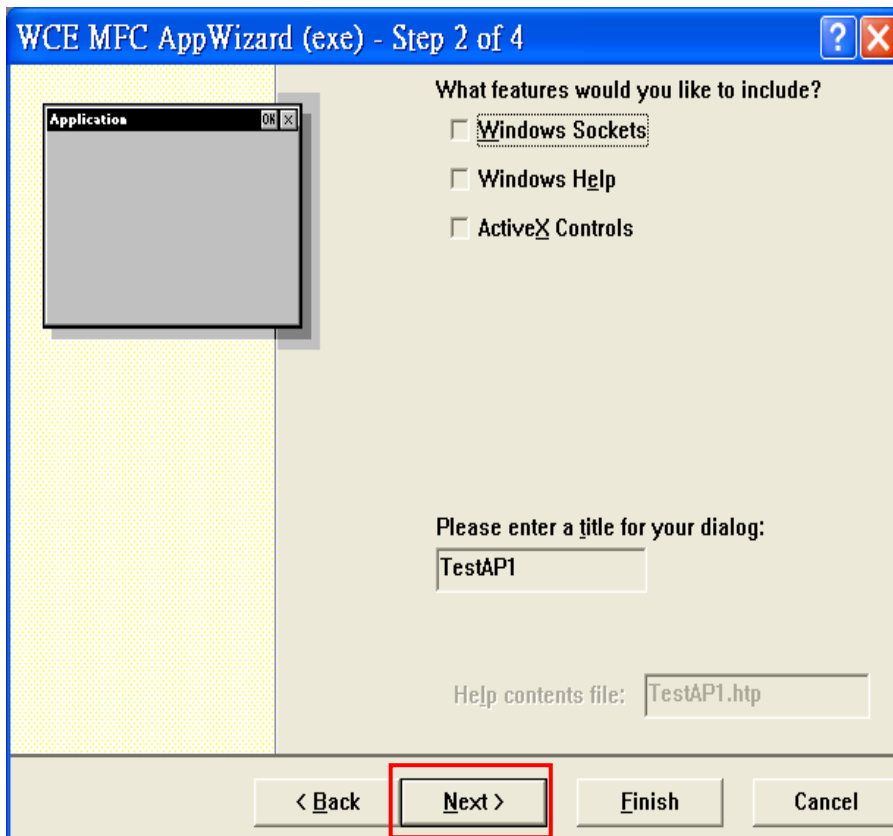
**STEP 3:** Select your application type, set up the location and name of your project and. Please select “WCE MFC AppWizard(exe)” as application type.



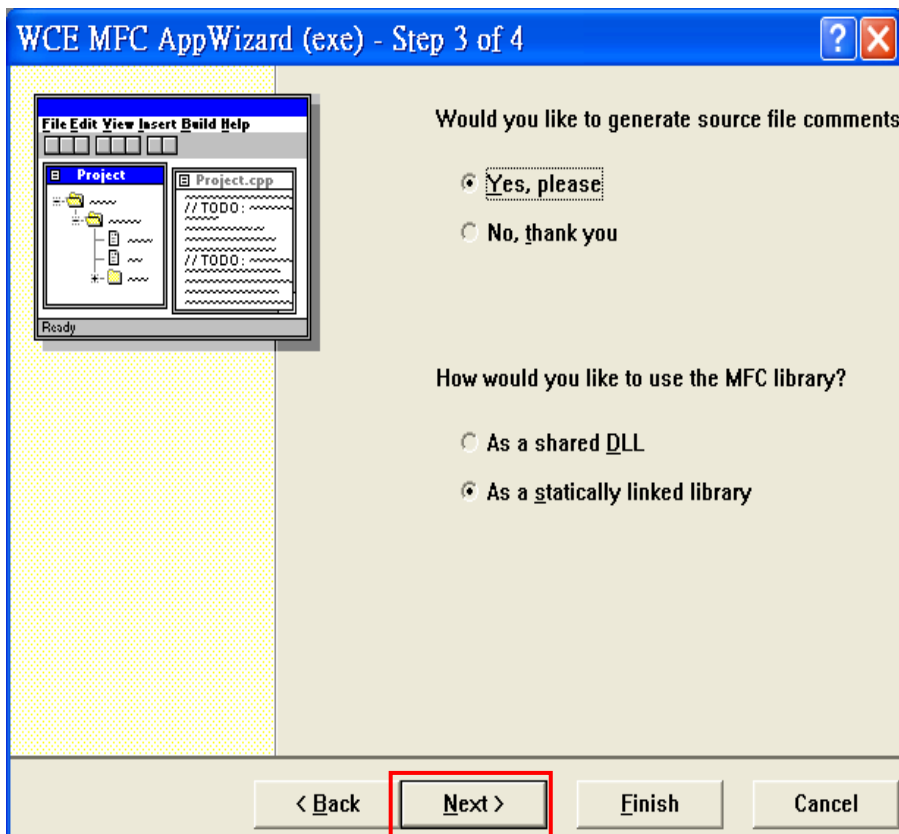
**STEP 4:** Select “Dialog based” and language setting. Click “Next” button.



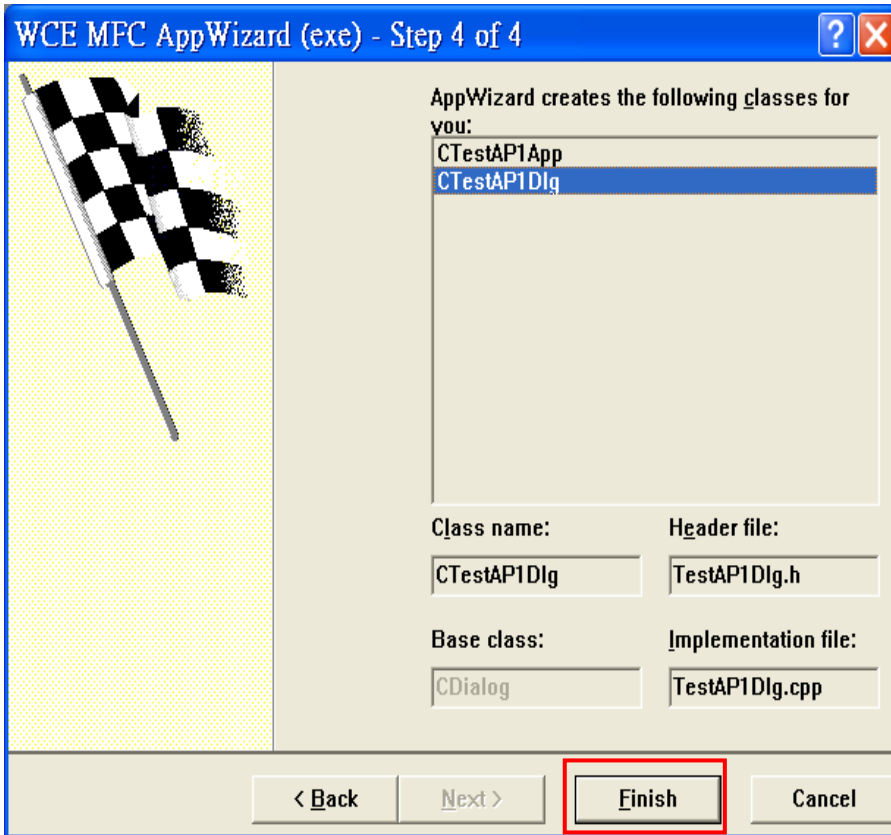
**STEP 5:** Click “Next” button.



**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

AT ÚÉ03A 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:

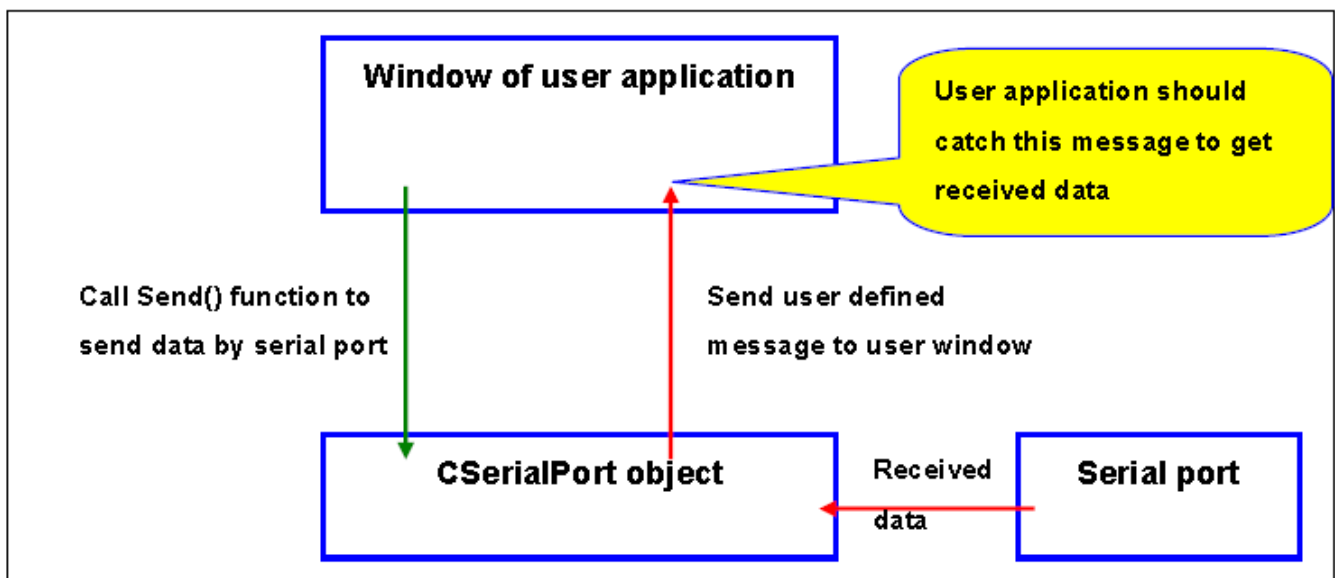
Name	Function	Comment
COM1:	GSM control port	Detail information could be found in section 4.4
COM4:	GPS control port	Detail information could be found in section 4.5
COM3:	Used by application program.	2-wire RS232
COM6:	Used by application program.	8-wire RS232
Debug port	For internal use only.	Can not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

## 4.3.2. Serial port control - CSerialPort class

Y ^ 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.

<b>Syntax</b>	CSerialPort( )
<b>Parameters</b>	None
<b>Return value</b>	None

**Open Function:** Open a serial port.

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

**Send Function:** Send specific data by this serial port.

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

**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.

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

**Close Function:** Close current serial port.

<b>Syntax</b>	BOOL Close ( );
<b>Parameters</b>	None
<b>Return value</b>	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.

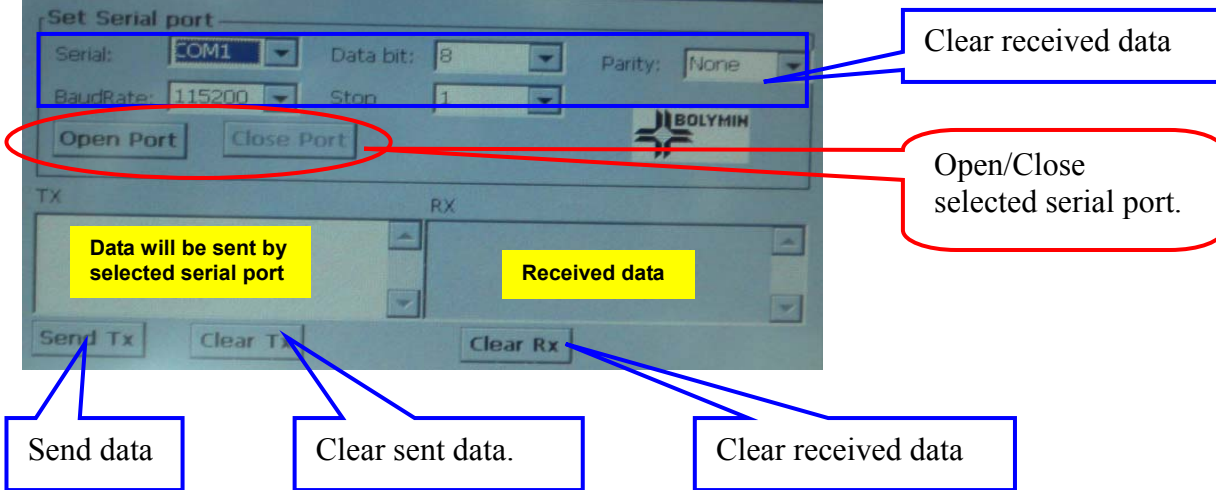
```
BEGIN_MESSAGE_MAP(CSerialPortDlg, CDialog)
//{{AFX_MSG_MAP(CSerialPortDlg)
ON_BN_CLICKED(IDC_OPEN_COM, OnOpenCom)
ON_BN_CLICKED(IDC_CLOSE_COM, OnCloseCom)
ON_BN_CLICKED(IDC_SEND, OnSend)
ON_BN_CLICKED(IDC_CLEAR_SEND, OnClearSend)
ON_BN_CLICKED(IDC_CLEAR_REC, OnClearRec)
ON_BN_CLICKED(IDC_CMD_TEST,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

Á

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);
    return TRUE;
}

```

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

```

/*****

```

**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();
```

Close current serial port.

```
    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);
```

Send data by current serial port.

```
    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 and delete CSerialPort object.

```
}
```

## 4.4. GSM Control

### 4.4.1. Overview

User application could communicate with GSM module of BETA903A by **COM1** with the following settings:

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

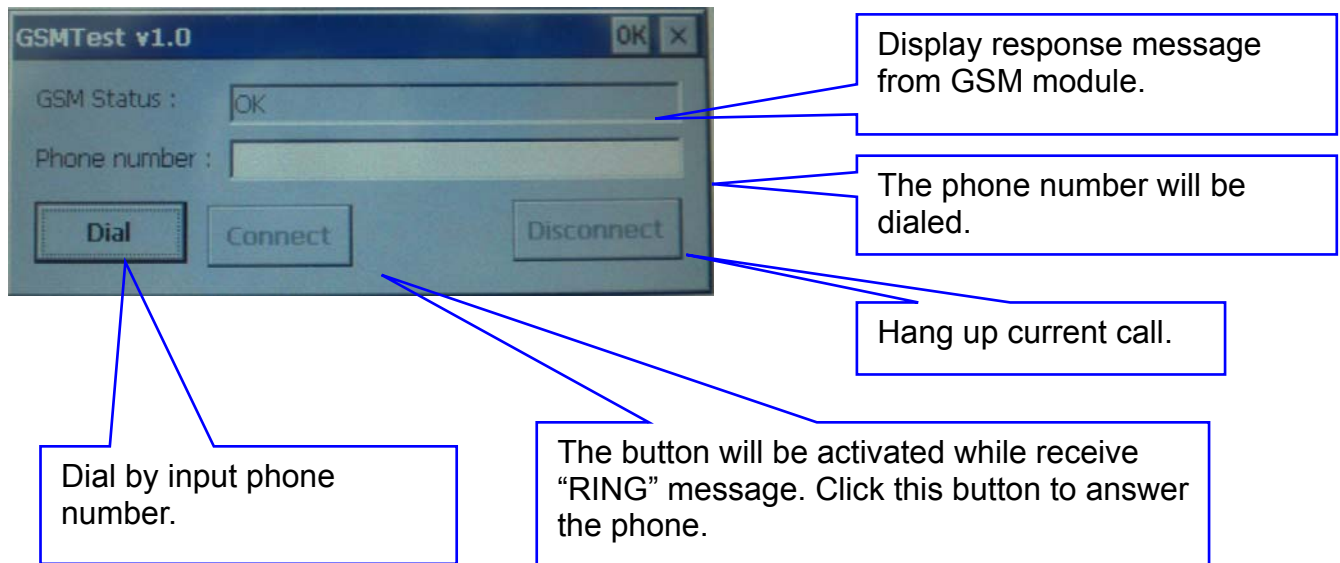
After opening COM1, user could send AT command to control the GSM module. Here is some AT commands used in our example application:

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. <b>The audio mode should be set to 2 for GSM module of BETA903A.</b>

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

## 4.4.2. Example code

Bolymin 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: pGSMTTestDlg.cpp  
const UINT WM_CMD_REVMMSG = WM_USER+1;  
BEGIN_MESSAGE_MAP(CPGSMTTestDlg, CDialog)  
    {{{AFX_MSG_MAP(CPGSMTTestDlg)  
        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()
```

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

```
// CPGSMTTestDlg message handlers
```

```
BOOL CPGSMTTestDlg::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 CPGSMTTestDlg::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;  
}
```

Open COM1 to  
communicate with GSM  
module.

Send "Echo off" command to  
GSM module.

```
CString temp_str;
```

```
// Echo off
```

```
temp_str = _T("ATE0\r");
```

```
SendATCmd(temp_str);
```

```
m_bSwitchAudioChannel = FALSE;
```

```
m_bConnectCall = FALSE;
```

```
// Load DLL of GPIO control functions of 903A
```

```
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 CPGSMTTestDlg::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 CPGSMTTestDlg::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_pSetGPOOutput(GO_AMP_SWITCH, FALSE);
    m_btnDisconnect.EnableWindow(FALSE); // Disable "DisConnect" button */
    m_btnConnect.EnableWindow(FALSE); // Disable "Connect" button */
}

UpdateData(FALSE);
return 0;
}

void CPGSMTTestDlg::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 CPGSMTTestDlg::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 */
```

```
}
```

```
void CPGSMTTestDlg::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 */
```

```
}
```

```
void CPGSMTTestDlg::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);
```

```
}
```

Hang up current call and switch amplifier to audio.

Setup a call by input phone number.

Switch amplifier to GSM function

## 4.5. GPS Message Translator

The GPS module of BETA903A 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 provided by Bolymin. 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.

<b>Syntax</b>	CGPSTranslator ( );
<b>Parameters</b>	None
<b>Return value</b>	None

**Translate Function:** GPS message translation function.

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

**IsGPSFixed Function:** Check if current GPS position fixed..

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

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

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

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

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

**GetGSADData Function:** Get data of the last translated GSA message..

<b>Syntax</b>	void GetGSADData ( stGSADData *data_ptr );
<b>Parameters</b>	data_ptr                      memory pointer of GSA data.
<b>Return value</b>	None

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

<b>Syntax</b>	void GetGLLData ( stGLLData *data_ptr );
<b>Parameters</b>	data_ptr                    memory pointer of GLL data.
<b>Return value</b>	None

**GetGGADData Function: Get data of the last translated GGA message..**

<b>Syntax</b>	void GetGGADData ( stGGADData *data_ptr );
<b>Parameters</b>	data_ptr                    memory pointer of GGA data.
<b>Return value</b>	None

**GetRMCDData Function: Get data of the last translated RMC message..**

<b>Syntax</b>	void GetRMCDData ( stRMCDData *data_ptr );
<b>Parameters</b>	data_ptr                    memory pointer of RMC data.
<b>Return value</b>	None

## 4.5.2. Data structure of GPS data

### Data structure: stRMCDData

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	cGroundSpeedUnitKnot	Unit of previous field, N=Knots.
double	dGroundSpeedInKM	Speed over ground in km/hour.
char	cGroundSpeedUnitKM	Unit of previous field, K=Kilometers per hour.
char	cMode	Mode indicator. (A=Autonomous, D=Differential, E=Estimated, N=Data not valid)

### Data structure: stGGADData

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	cAltitudeUnit	Unit of previous field. M=meter.
double	dHOG	Height of geoid above WGS84 ellipsoid.
char	cHOGUnit	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: stGSData

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 Vehicles(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

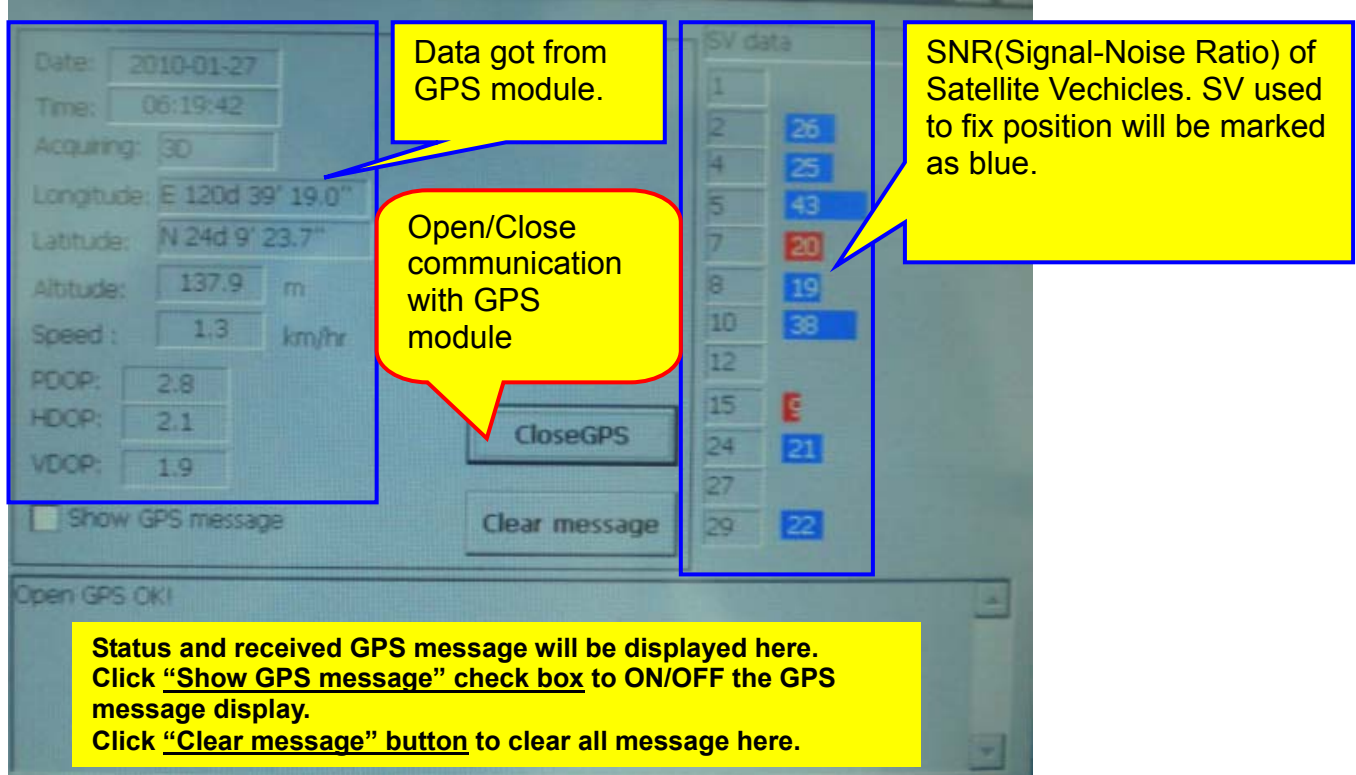
Data type	Data name	Description
int	iSVCount	Number of Satellites in view.
stSVData	SVData[12]	Data of Satellites in view.

#### Data structure: stSVData

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

### 4.5.3. Example code

Bolymin 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();
```

Create object of 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;
```

```

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
        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);
}

```

Open COM4 to communicate with GPS module.

```
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;
}

```

Send current received message to GPSTranslator

m\_pGPSTranslator->Translate(&m\_cMsgBuf[0], m\_iCurBufIndex);

```

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

    CDialog::OnTimer(nIDEvent);
}

```

Implement function used to display current GPS data  
 /\*\*\*\*\*

```

void CPGPSTestDlg::UpdateScreen()
{

```

```

    stRMCDData data_RMC;
    stGSADData data_GSA;
    stGGADData data_GGA;
    stVTGData data_VTG;
    stGSVData data_GSV;

```

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
```

```
{
```

```
  tmp_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);
```

Get current GGA data

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

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

Get current VTG data

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

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

```
}
```

```
// Update SV status
```

```
CRect rect;
```

Get current GSV data

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

```
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 of BETA903A

Bolymin provides a DLL file “pGPIO\_903A.dll” to control the General Purpose Input and Output(GPIO) signal. User could read current value of all GPIO of BETA903A 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);
```

Load “pGPIO\_903A.dll” and get the address of GPIO control functions.

```
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);
```

Execute GPIO control functions.

## 4.6.2. GPIO control functions for BETA903A

**GetGPInput Function:** Get current status of specified GPIO.

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

**SetGPOutputFunction:** Set value of specified GP Output.

<b>Syntax</b>	<pre>void SetGPOutput (     int gpio_index,     BOOL value );</pre>
<b>Parameters</b>	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.
<b>Return value</b>	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 : <b>HIGH</b> .
GO_LCD_POWER_ENABLE	LCD power control. Default value : <b>LOW</b> .
GO_EARPHONE_SWITCH	Earphone switch. Default value : <b>HIGH</b> . Set LOW when earphone is used for GSM function.
GO_GSM_MIC_SWITCH	Reserved for internal use.
GO_AMP_SWITCH	Amplifier switch. Default vale: <b>LOW</b> . Set HIGH when amplifier is used for GSM function.

## 4.6.4. Keypad control

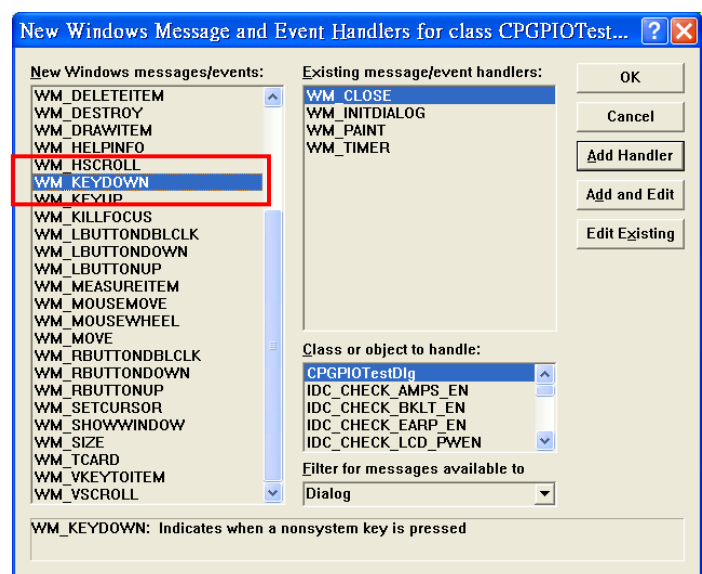
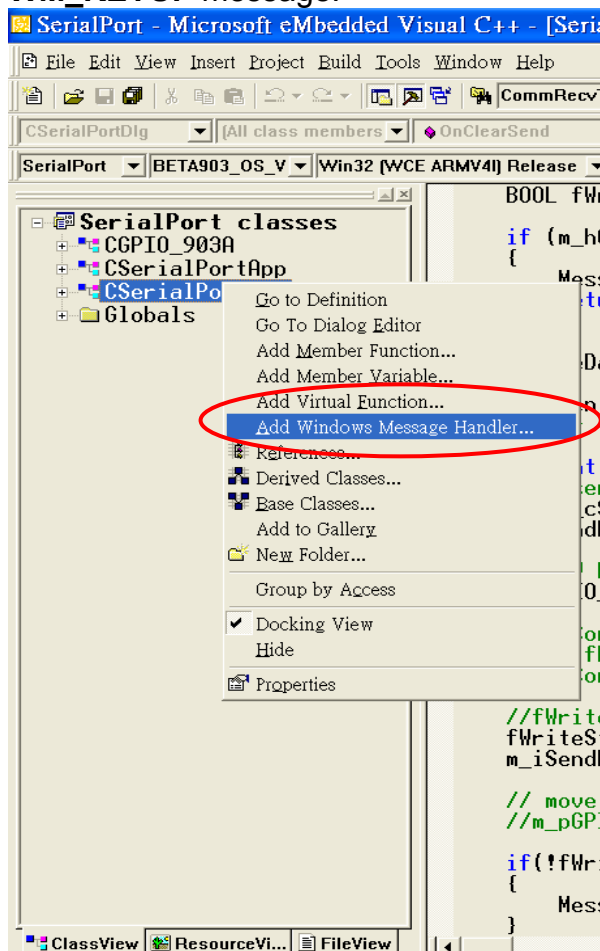
The keypad of BETA903A supports 8 user-defined buttons. The buttons of keypad of BETA903A will map to **F11~F18** of keyboard. To learn which button of keypad is pressed, you may catch WM\_KEYDOWN or WM\_KEYUP message in your program and add the process codes. Here is a step guide:

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

```
// Override this function to catch key message
BOOL CPGPIOTestDlg::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 T ÔÚË903A User Manual >