V2400A Series Expansion Modules User's Manual

Edition 2.1, April 2017

www.moxa.com/product

V2400A Series Expansion Modules User's Manual

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Trademarks

The MOXA logo is a registered trademark of Moxa Inc.
All other trademarks or registered marks in this manual belong to their respective manufacturers.

Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document as is, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Technical Support Contact Information

www.moxa.com/support

Moxa Americas
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

Moxa China (Shanghai office)
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-21-5258-5505

Moxa Europe
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

Moxa Asia-Pacific
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

Moxa India
Tel: +91-80-4172-9088
Fax: +91-80-4132-1045

Table of Contents

1

Introduction

Moxa's EPM series expansion modules, which include modules with serial ports, a wireless/GPS card, a digital input/output channel card, a CAN bus card, and a 2-slot mini PCIe card, work with Moxa's V2426A embedded computer, giving end-users the ability to set up and expand a variety of industrial applications.

The following topics are covered in this chapter:

- **Overview**
- **Package Checklist**
- **Available Modules**
- **EPM Module Specifications**
  - EPM-3032 Specifications
  - EPM-3112 Specifications
  - EPM-3438 Specifications
  - EPM-DK02 Specifications
  - EPM-DK03 Specifications

Overview

Moxa's EPM series modules provide expansion options for the V2426A embedded computer. These modules may provide serial ports, combined Wi-Fi/GPS, a digital I/O channels, CAN bus interface, and a 2-slot mini PCIe card, giving end-users the ability to set up V2400 computers for a wide variety of industrial applications.

Package Checklist

Each package ships with a single EPM expansion module and a quick installation guide

NOTE: Please notify your sales representative if the module is damaged en route.

Available Modules

- EPM-3032: 2 isolated RS-232/422/485 ports, DB9 connectors
- EPM-3112: 2 isolated CAN ports with DB9 connectors
- EPM-3438: 8+8 DI/DO, with 3 KV digital isolation protection, 2 KHz counter
- EPM-DK02: 2-slot Mini PCIe expansion module
- EPM-DK03: GPS + 2-slot Mini PCIe expansion module

EPM Module Specifications

EPM-3032 Specifications

Serial Interface

Serial Standards: 2 RS-232/422/485 ports, software-selectable (DB9 male)

Isolation: 2-KV digital isolation

Serial Communication Parameters

Data Bits: 5, 6, 7, 8

Stop Bits: 1, 1.5, 2

Parity: None, Even, Odd, Space, Mark

Flow Control: RTS/CTS, XON/XOFF, ADDC® (automatic data direction control) for RS-485

Baudrate: 50 bps to 921.6 Kbps (non-standard baudrates supported; see user's manual for details)

Serial Signals

RS-232: TxD, RxD, DTR, DSR, RTS, CTS, DCD, GND

RS-422: TxD+, TxD-, RxD+, RxD-, GND

RS-485-4w: TxD+, TxD-, RxD+, RxD-, GND

RS-485-2w: Data+, Data-, GND

Physical Characteristics

Weight: 137 g

Dimensions: 104 x 121 x 34 mm (4.09 x 4.76 x 1.34 in)

Environmental Limits

Operating Temperature: -40 to 70°C (-40 to 158°F)

EPM-3112 Specifications

CAN bus Communication

Interface: 2 optically-isolated CAN 2.0 A/2.0 B compliant ports
CAN Controller: Phillips SJA1000T
Signals: CAN-H, CAN-L
Isolation: 2 KV digital isolation
Speed: 1 Mbps
Connector Type: DB9 male

Physical Characteristics

Weight: 127 g
Dimensions: 104 x 121 x 34 mm (4.09 x 4.76 x 1.34 in)

Environmental Limits

Operating Temperature: -25 to 55°C (-13 to 131°F)

EPM-3438 Specifications

Digital Input

Input Channels: 8, source type
Input Voltage: 0 to 30 VDC at 25 Hz
Digital Input Levels for Dry Contacts:
• Logic level 0: Close to GND
• Logic level 1: Open
Digital Input Levels for Wet Contacts:
• Logic level 0: +3 V max.
• Logic level 1: +10 V to +30 V (Source to DI)
Counter Frequency: 2 KHz (DI0 only)
Connector Type: 10-pin screw terminal block (8 DI points, DI Source, GND)
Isolation: 3 KV optical isolation

Digital Output

Output Channels: 8, sink type, 0 to 30 VDC
Output Current: Max. 200 mA per channel
On-state Voltage: 24 VDC nominal, open collector to 30 VDC
Connector Type: 9-pin screw terminal block (8 DO points, GND)
Isolation: 3 KV optical isolation

Physical Characteristics

Weight: 120 g
Dimensions: 104 x 121 x 34 mm (4.09 x 4.76 x 1.34 in)

Environmental Limits

Operating Temperature: -40 to 70°C (-40 to 158°F), EN 50155 Class TX

EPM-DK02 Specifications

PCI Express Mini Slot

Interface:
Slot 1: PCI Express V1.0 (one lane) / USB 2.0
Slot 2: USB 2.0
USB 2.0 Bus SIM Card Holder: Reserved for cellular applications

Physical Characteristics

Weight: 125 g

Environmental Limits

Operating Temperature: -25 to 55°C (-13 to 131°F), EN 50155 Class T1

EPM-DK03 Specifications

PCI Express Mini Slot

Interface 1: PCI Express V1.0 (one lane) / USB 2.0

Interface 2: USB 2.0

USB 2.0 Bus SIM Card Holder: Reserved for cellular applications

GPS Interface

Acquisition:

- Cold starts: 28s
- Warm starts: 28s
- Aided starts: 1s
- Hot starts: 1s

Sensitivity:

- Tracking: -160 dBm
- Reacquisition: -160 dBm
- Cold starts: -147 dBm

Timing accuracy:

- RMS: 30 ns
- 99%: <60 ns
- Granularity: 21 ns

Accourcie:

- Position: 2.5 m CEP
- SBAS: 2.0 m CEP

Protocol: NMEA; UBX binary, 5 Hz max. update rate (ROM version)

Time Pulse: 0.25 Hz to 1 KHz

Velocity Accuracy: 0.1 m/s

Heading Accuracy: 0.5 degrees

A-GPS: Supports AssistNow Online and AssistNow Offline, OMA SUPL compliant

Velocity Limit: 500 m/s (972 knots)

Physical Characteristics

Weight: 125 g

Dimensions: 104 x 121 x 34 mm (4.09 x 4.76 x 1.34 in)

Environmental Limits

Operating Temperature: -25 to 55°C (-13 to 131°F), EN 50155 Class T1

2

Hardware Introduction

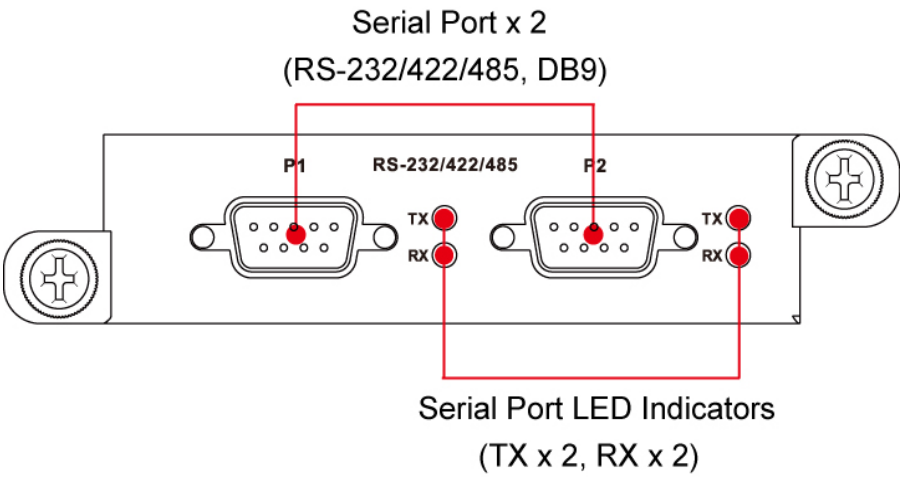
The EPM series expansion modules are designed to work with Moxa's V2426A embedded computer. By providing different modules with different connectors, the EPM series offers the greatest flexibility and convenience for users who would like to easily establish industrial applications that require different communication interfaces.

The following topics are covered in this chapter:

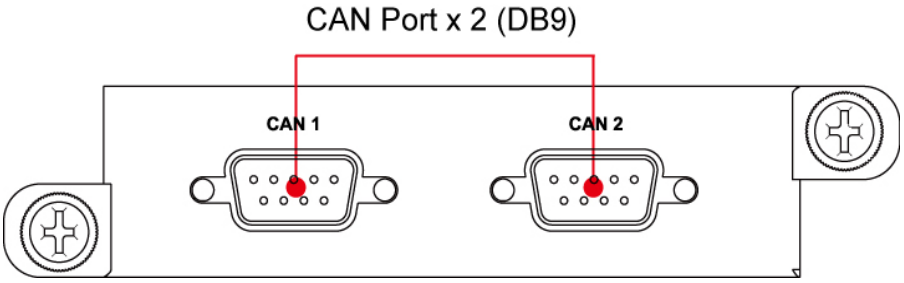
- **Appearance**
  - EPM-3032
  - EPM-3112
  - EPM-3438
  - EPM-DK02
  - EPM-DK03
- **Dimensions**

Appearance

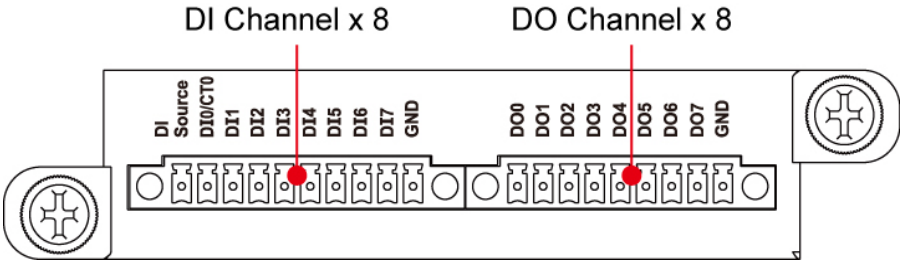
EPM-3032



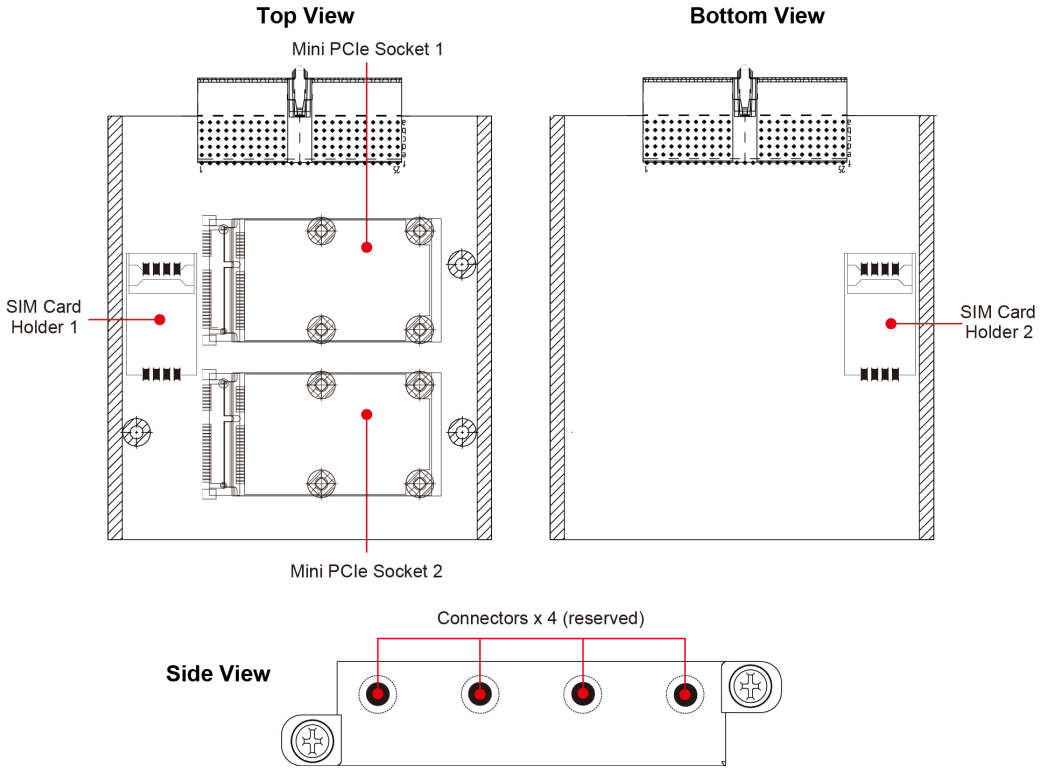
EPM-3112



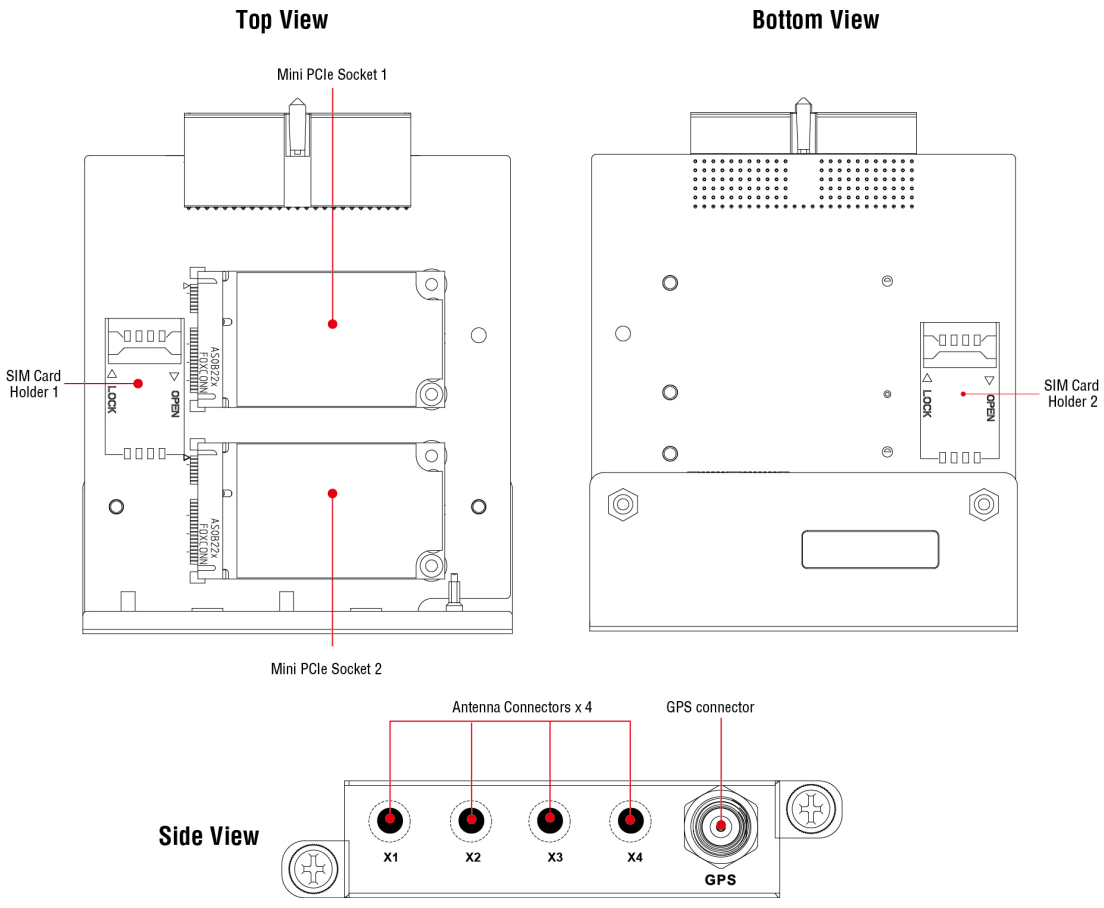
EPM-3438



EPM-DK02

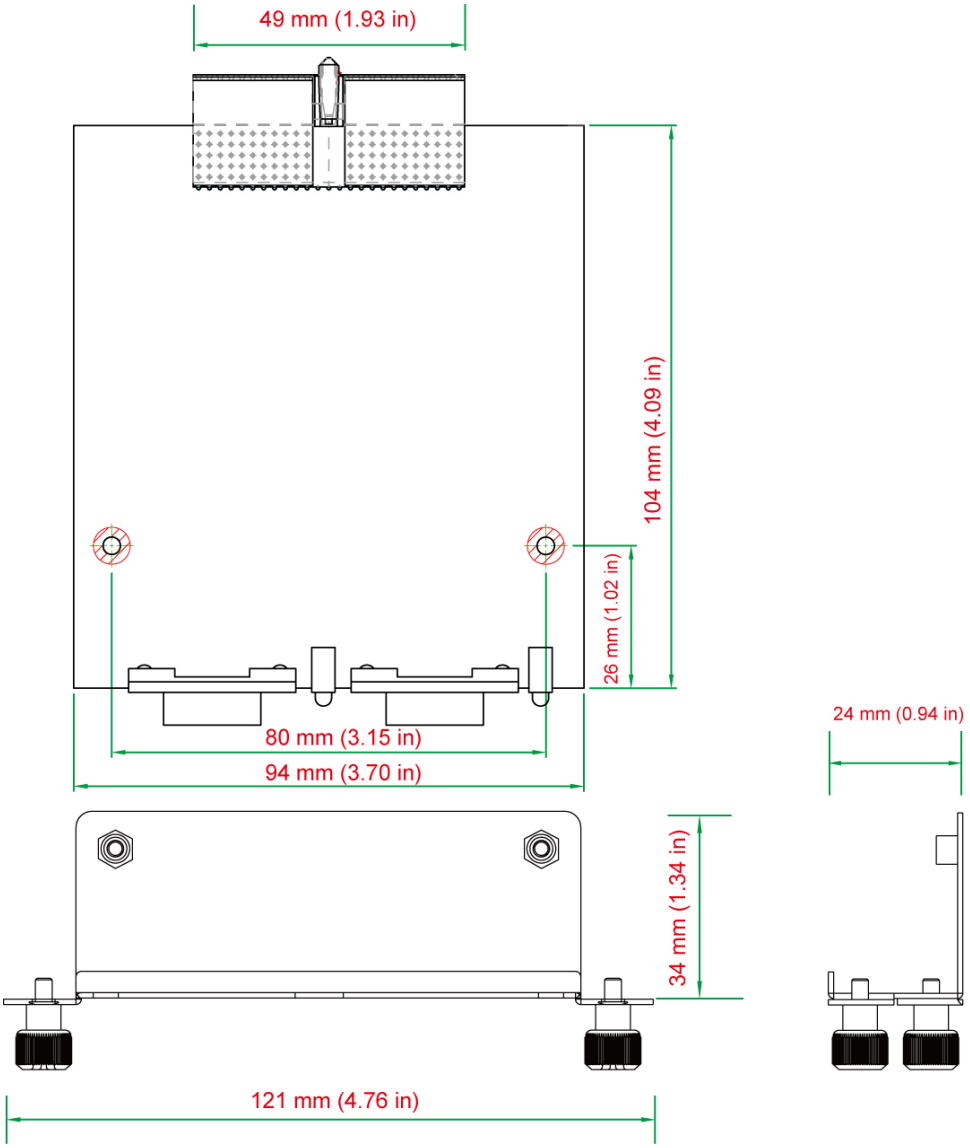


EPM-DK03



Dimensions

The dimensions of all the modules are as indicated below:



3

Hardware Connection Description

In this chapter, we explain how to connect the V2400A series expansion modules.

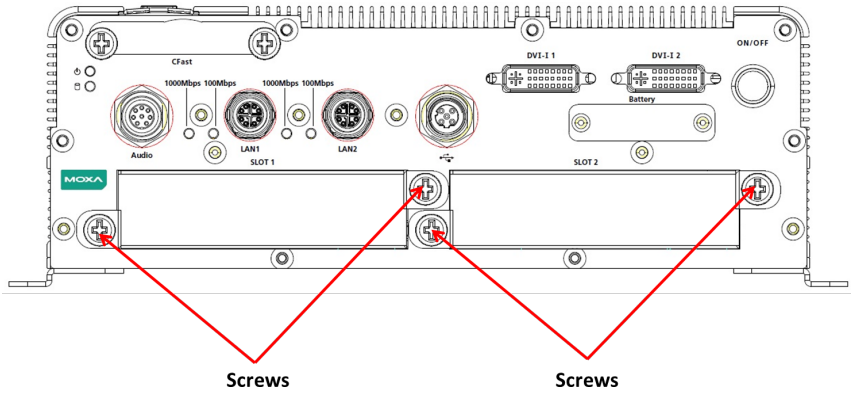
The following topics are covered in this chapter:

- **Installing the EPM Expansion Modules**
- **Connecting Data Transmission Cables**
  - EPM-3032 Serial Port Module
  - EPM-3438 DI/DO Module
  - EPM-3112 CAN Bus Module
  - EPM-DK02 Module: 2 mini PCIe Sockets
- **Installing Cellular/Wi-Fi Modules on the EPM-DK02/EPM-DK03**
  - Configuring Power Controls on Socket 1
  - Installing a Cellular Module
  - Arranging Cables on the Wi-Fi or Cellular Modules

Installing the EPM Expansion Modules

The EPM series expansion modules are designed to work with Moxa's V2426A embedded computer. To insert an expansion module into a slot on the embedded computer:

1. Remove a slot cover by unfastening the screws.



2. Insert the module into the slot.
3. When finished, tighten the screws to hold the module in place.

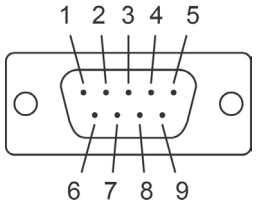
Connecting Data Transmission Cables

In this section we explain how to connect devices to the EPM modules.

EPM-3032 Serial Port Module

Use a serial cable to plug your serial device into the module's serial port. Serial ports 1 and 2 are provided with male DB9 connectors and can be configured for RS-232, RS-422, or RS-485 communication by software. The pin assignments are shown in the following table:

DB9 Male Port

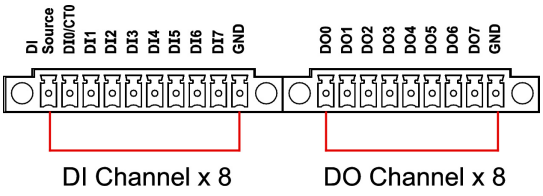


RS-232/422/485 Pinouts

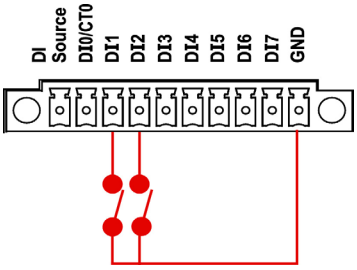
Pin	RS-232	RS-422	RS-485 (4-wire)	RS-485 (2-wire)
1	DCD	TxDA(-)	TxDA(-)	–
2	RxD	TxDB(+)	TxDB(+)	–
3	TxD	RxDB(+)	RxDB(+)	DataB(+)
4	DTR	RxDA(-)	RxDA(-)	DataA(-)
5	GND	GND	GND	GND
6	DSR	–	–	–
7	RTS	–	–	–
8	CTS	–	–	–

EPM-3438 DI/DO Module

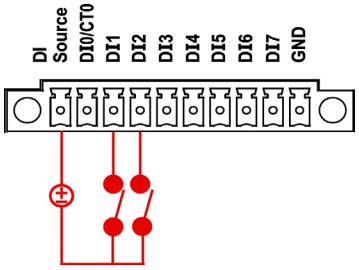
The EPM-3438 module comes with 8 digital input channels and 8 digital output channels. See the following figures for pin definitions and wiring methods.



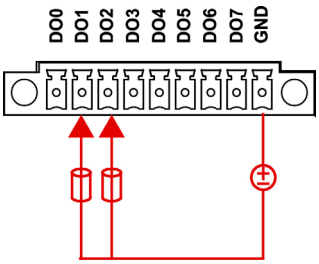
Digital Input
Dry Contact Wiring



Digital Input
Wet Contact Wiring



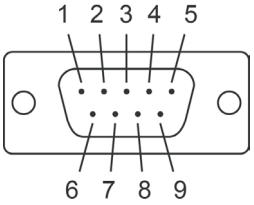
Digital Output Wiring



EPM-3112 CAN Bus Module

The EPM-3112 offers two CAN bus ports with DB9 male connectors. Use a cable to plug your CAN device into the module's serial port. The pin assignments are shown in the following table:

DB9 Male



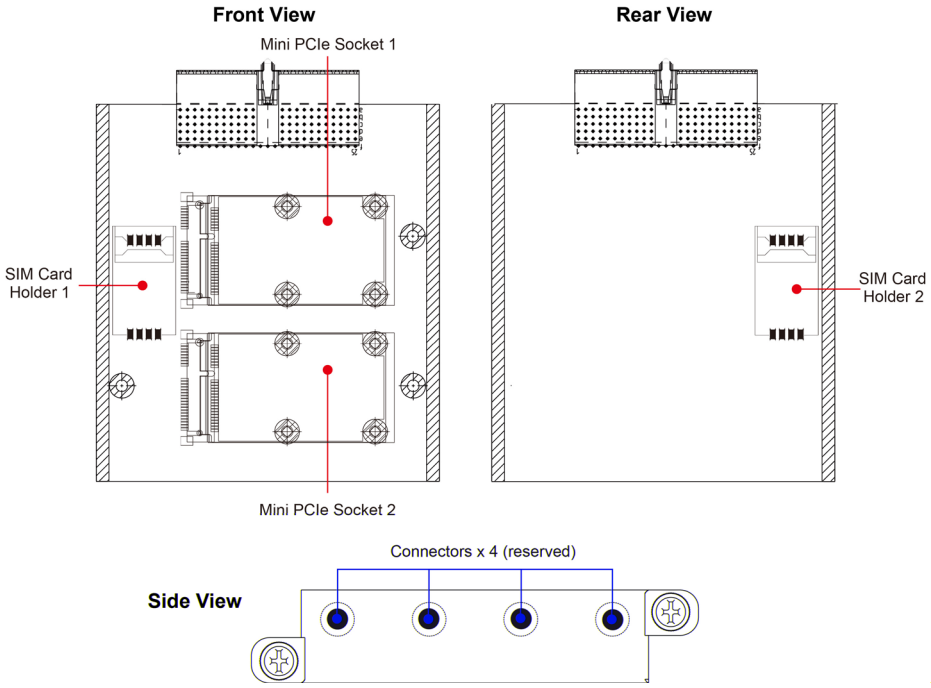
CAN bus Pinouts

PIN	CAN
1	–
2	CAN-L
3	–
4	–
5	–
6	–
7	CAN-H
8	–
9	–

EPM-DK02 Module: 2 mini PCIe Sockets

The EPM-DK02 has two mini PCIe sockets that may be used for cellular communications expansion. The figures below show the slots' specific locations. Note that while both sockets provide a mini PCIe interface, socket one supports either mini PCIe or USB 2.0 signals, whereas socket two only supports USB 2.0 signals.

Connect the cellular module to the mini PCIe socket, and insert the SIM card into the SIM card holder. Be sure to tighten the screw in the screw holder so that the module will be firmly installed. Note that the second SIM card holder is located on the back of the module. If you need to connect antennas, use the connectors on the side panel.



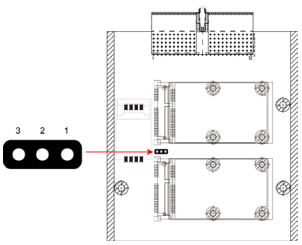
Installing Cellular/Wi-Fi Modules on the EPM-DK02/EPM-DK03

The EPM-DK02/03 has two mini PCIe sockets that allow users to insert two mini PCIe cards for cellular or Wi-Fi communication. Note that while both sockets provide a mini PCIe interface, socket 1 supports either mini PCIe or USB 2.0 signals, whereas socket 2 only supports USB 2.0 signals.

Users may purchase the cellular and Wi-Fi modules for the EPM-DK02/03 separately. The instructions for installing the cellular and Wi-Fi modules are specified in the following sections. To make it more convenient to install the antenna, install the module at Socket 2 first.

Configuring Power Controls on Socket 1

To use socket 1 as a USB interface, the user must allow the platform's general-purpose input/output (GPIO) to control the power; in contrast, for the PCIe interface the power must be constantly on. These power controls must first be set at the hardware level, using the jumper shown in the figure at right.

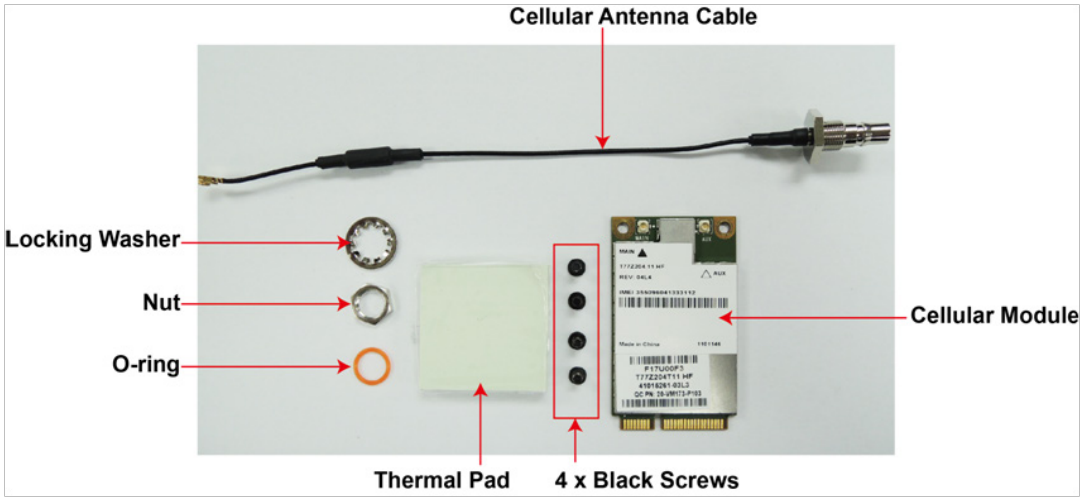


1. To enable GPIO control of power for a USB interface, place the jumper on pins 1 and 2.
2. To disable power controls for an always-on PCIe interface, place the jumper on pins 2 and 3.

NOTE This jumper configuration is for socket 1 only.

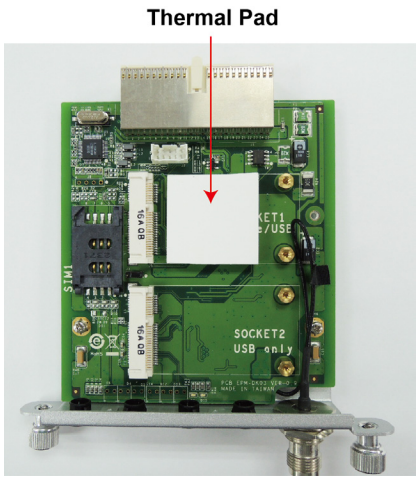
Installing a Cellular Module

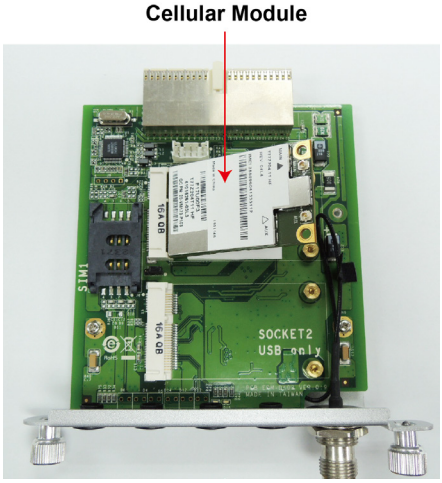
The cellular accessory package includes a cellular module, a cellular antenna cable, four black screws, a thermal pad, a locking washer, a nut, and an O-ring. In addition, a printed quick installation guide is also included in the kit.



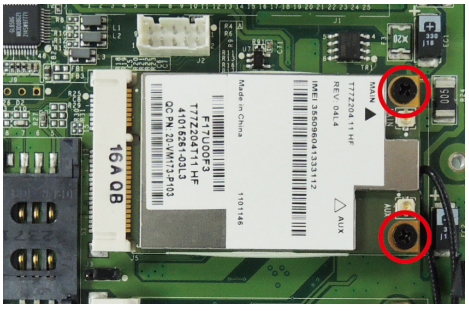
See the following steps to install the cellular module.

1. Remove the membranes on both sides of the thermal pad, and then place the thermal pad on the socket. When finished, insert the cellular module into the socket.

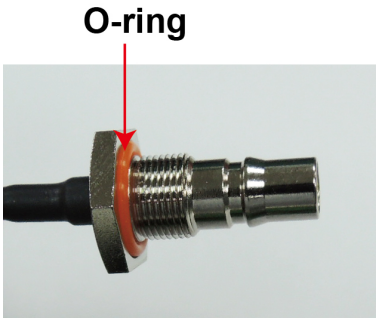


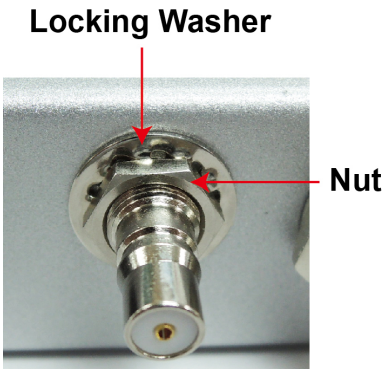


2. Fasten the module with 2 black screws. Connect the antenna cable on the module. Make sure that the cable has been securely fastened. See the following figures for the specific locations of the screws and the antenna cable connector.

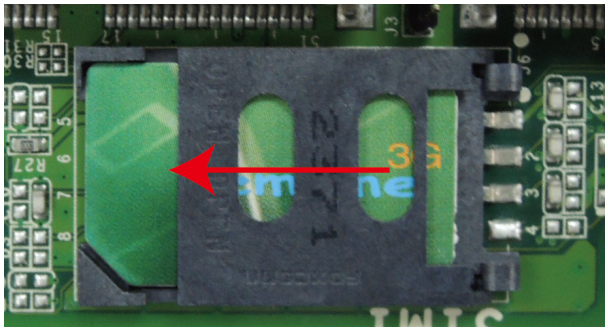


3. Insert the connector of the antenna cable into the O-ring. Remove the black cover from the antenna hole in the front side of the module, and then insert the connector. Insert the locking washer first, and then fasten the nut to secure the connector (see figure on next page for detailed pictures).





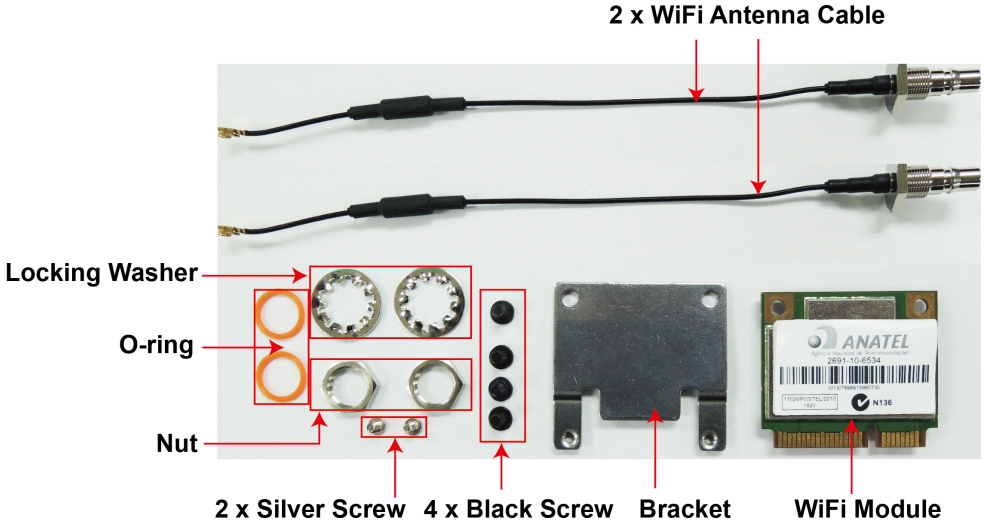
4. When finished, connect the cellular antenna to the connector. We recommend installing the antenna on either X2 or X3.
5. Next, insert the SIM card for the cellular module. Pull up the SIM card slot and insert the SIM card. When finished, replace the holder and slide the slot towards the catch to fasten the holder



6. You can use the same procedure to install another cellular module on another socket. The second SIM card holder is located on the back of the module
7. To complete the installation, please continue reading in to the section below, **Arranging Cables on the Wi-Fi or Cellular Modules**.

Installing a Wi-Fi Module

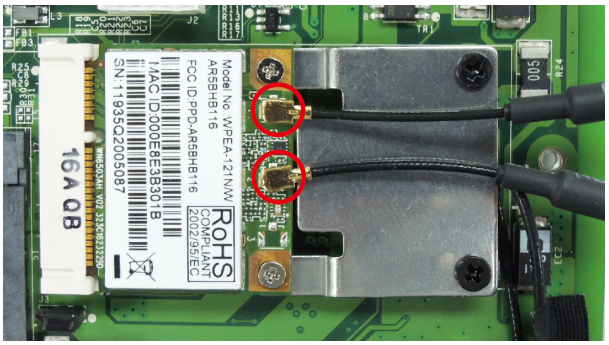
Moxa's Wi-Fi module accessory package includes a Wi-Fi module, a bracket, two Wi-Fi antenna cables, four black screws, two silver screws, a locking washer, a nut, and an O-ring. A printed quick installation guide is also included.



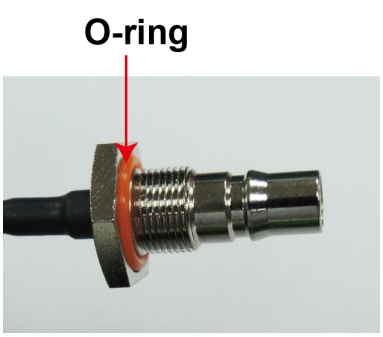
To install a Wi-Fi module:

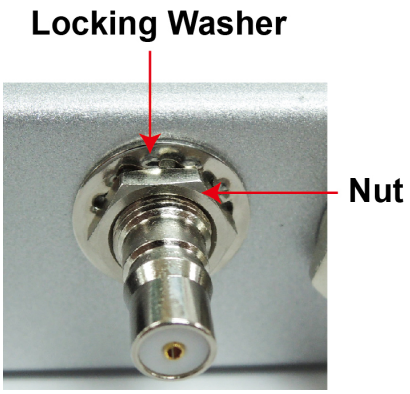
1. Use the two silver screws to fasten the stabilization bracket to the Wi-Fi module. Make sure you connect the bracket in the correct direction: the two tongues which are attached to the Wi-Fi module should, after installation, be positioned under the card (refer to the figure below for clarification). Insert the Wi-Fi module into Socket 1, and then fasten with the bracket into place using the two black screws. Please note that Wi-Fi module can only be inserted in Socket 1. See the following figures for details.

2. Connect the two antenna cables on the module. Make sure that these cables are securely fastened on the module.



3. Insert the connector of the antenna cable into the O-ring. Remove the black cover from the antenna hole in the front side of the module, and then insert the connector. Place the locking washer on the connector, and then fasten the nut to secure the connector.





4. For best performance, the first connector should protrude from the X1 hole, and the second from the X4 hole.
5. To complete the installation, please continue reading in to the section below, **Arranging Cables on the Wi-Fi or Cellular Modules**.

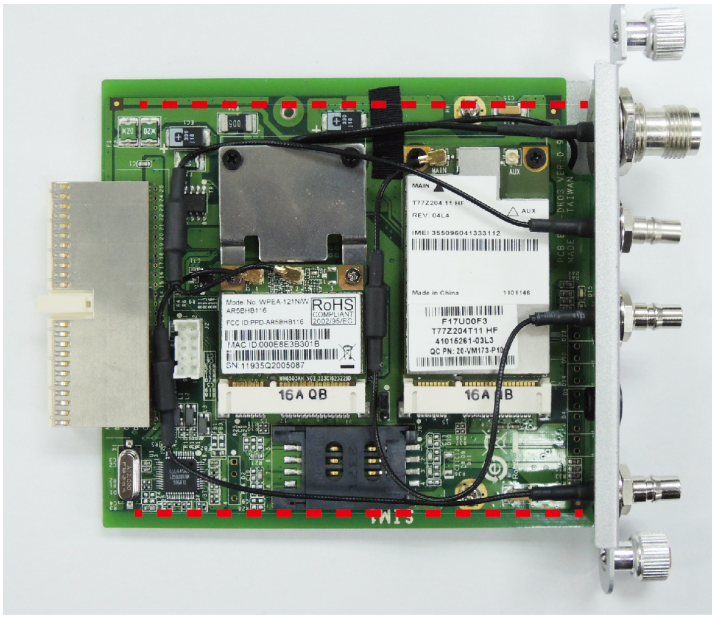
When finished, you may mount the DK03 module into one of Moxa's V2400A Series embedded computers

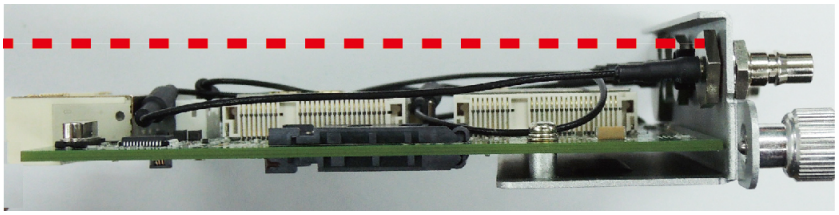
WARNING

The Wi-Fi module can only be inserted into socket 1. Do not attempt to force it into socket 2.

Arranging Cables on the Wi-Fi or Cellular Modules

After installing the cellular and Wi-Fi modules, make sure to arrange the cables properly within the chassis area. The proper arrangement is shown in the following photos.





4

Software Installation and Programming Guide

In this chapter we discuss software installation and programming guide for the EPM expansion modules 3032, DK02, and DK03.

The following topics are covered in this chapter:

- **Linux System Peripherals Programming Guide**
- **EPM-3032: Driver Installation**
  - EPM-3032 Programming Guide
- **EPM-3438 Driver Installation**
  - EPM-3438 Programming Guide
  - Implementing Timer Functions on Digital IO Ports
- **EPM-3112 CAN Bus Interface**
  - EPM-3112 Driver Installation
  - EPM-3112 Configuring the Socket CAN Interface
  - EPM-3112 Programming Guide
- **Installing the EPM-DK02 Kernel Module**
  - EPM-DK02 Kernel Module
  - Configuring Power Control
  - Wi-Fi Module
  - Cellular Module (PLS8-X R3 and Above)
- **Installing the EPM-DK03 Module**
  - Installing the GPS Test Clients
- **Windows System**
- **EPM-3032: Driver Installation**
  - EPM-3032: Configuring Serial Port Mode
  - EPM-3032: Changing the Software-Selectable UART Mode
- **EPM-3438: Driver Installation**
  - EPM-3438: Programming Guide
- **EPM-3112: Driver Installation**
  - EPM-3112: Programming Guide
- **EPM-DK02: Driver Installation**
  - EPM-DK02: Controlling Power
- **EPM-DK03: GPS Driver Installation**
  - EPM-DK03: GPS Module Configuration
- **Wi-Fi Module Driver Installation**
  - Wi-Fi Module Driver Configuration
- **LTE Module Driver Installation**
  - LTE Module Configuration
- **3G Module Driver Installation**
  - 3G Module Configuration
- **3G-GPS Module Driver Installation**
- **3G-GPS Module Driver Configuration**

Linux System Peripherals Programming Guide

EPM-3032: Driver Installation

The EPM-3032 may be accessed through the Linux console as a **tty** device node. The Moxa driver creates a special device node that is identified as a **ttyM*** device. The EPM-3032 device nodes are listed as **/dev/ttyM0** and **/dev/ttyM1**, or alternately as **/dev/ttyM8** and **/dev/ttyM9**. The UART API allows you to configure these device nodes for RS-232, RS-422, 4-wire RS-485, or 2-wire RS-485.

Upload the driver and tool packages to **/dev/shm**, a temporary file system on your computer.

```
root@Moxa:~# scp v2400a-mxser_1.0.0_amd64.deb moxa@192.168.3.127:/dev/shm
root@Moxa:~# scp v2400a-setinterface_1.0.0_amd64.deb moxa@192.168.3.127:/dev/shm
```

Install the package

```
root@Moxa:~# cd /dev/shm
root@Moxa: /dev/shm# dpkg -i v2400a-mxser_1.0.0_amd64.deb
Selecting previously unselected package v2400a-mxser.
(Reading database ... 48005 files and directories currently installed.)
Unpacking v2400a-mxser (from v2400a-mxser_1.0.0_amd64.deb) ...
Setting up v2400a-mxser (1.0.0) ...
root@Moxa: /dev/shm# dpkg -i v2400a- setinterface _1.0.0_amd64.deb
(Reading database ... 48011 files and directories currently installed.)
Unpacking v2400a-setinterface (from v2400a-setinterface_1.0.0_amd64.deb) ...
Setting up v2400a-setinterface (1.0.0) ...
root@Moxa: /dev/shm# reboot
```

The EPM-3032 driver, **mxser.ko** loads automatically when the system boots up.

To uninstall the driver, use the following command:

```
root@Moxa:~# dpkg --purge v2400a-setinterface
root@Moxa:~# dpkg --purge v2400a-mxser
```

EPM-3032 Programming Guide

Example 1: Setting the Modulation Rate/Baudrate

```
#define MOXA                0x400
#define MOXA_SET_SPECIAL_BAUD_RATE     (MOXA+100)
#define MOXA_GET_SPECIAL_BAUD_RATE     (MOXA+101)
#include    <termios.h>
    struct termios  term;
    int         fd, speed;
    fd = open("/dev/ttyM0", O_RDWR);
    tcgetattr(fd, &term);
    term.c_cflag &= ~(CBAUD | CBAUDEX);
    term.c_cflag |= B4000000;
    tcsetattr(fd, TCSANOW, &term);
    speed = 115200;
    ioctl(fd, MOXA_SET_SPECIAL_BAUD_RATE, &speed);
```

Example: Viewing the Modulation Rate/Baudrate

```
#define MOXA                0x400
#define MOXA_SET_SPECIAL_BAUD_RATE     (MOXA+100)
#define MOXA_GET_SPECIAL_BAUD_RATE     (MOXA+101)
#include    <termios.h>
    struct termios  term;
    int         fd, speed;
    fd = open("/dev/ttyM0", O_RDWR);
    tcgetattr(fd, &term);
    if ( (term.c_cflag & (CBAUD|CBAUDEX)) != B4000000 ) {
        // On this line, you may insert a standard baud rate
    } else {
        ioctl(fd, MOXA_GET_SPECIAL_BAUD_RATE, &speed);
    }
```

ATTENTION

The maximum baudrate for the serial ports is 921,600 bps. The serial port expansion module supports modulation rates of up to 921,600 baud. Standard baudrates are: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, and 921600. To configure the above code for a standard baudrate connection, simply insert the number on the indicated line.

If you use **stty** to get interface stats from a connection configured for a non-standard baud, the system will return a rate of 0.

Modulation Rate Inaccuracy

If you want to use to a non-standard baudrate, use the following equation to calculate the baudrate tolerance to minimize signal errors:

*** Inaccuracy = (Target Baudrate – 921600/(Divisor + (ENUM/8))) / Target Baudrate * 100%

The variables in the above equation are described below:

Divisor = Integer part of [921,600 / Target Baudrate]
ENUM = 8 * (921600 / Target baudrate - Divisor) (Round up or down)

E.g., to calculate the inaccuracy for 500,000 bps:

Divisor = 1, ENUM = 7,
Inaccuracy = 1.7%

Note: For reliable performance, the inaccuracy should be less than 2%

Configuring Serial Port Mode

Use the **setinterface** command to retrieve the parameters of the serial port configuration. The usage is **$:~# setinterface [device node] [interface option]**. The **device node** is the tty device to be configured. For the serial ports, Moxa uses a proprietary driver whose device nodes are identified with the marker **M**. Serial ports 1 and 2 (respectively) on card 1 are referred to as **ttyM0** and **ttyM1**, and **ttyM8** and **ttyM9** refer to ports 1 and 2 (respectively) on the second card. The **interface option** is a number between 0 and 4 that will determine what serial interface should be configured for the port in question. For example,

```
$:~# setinterface /dev/ttyM0 0
```

This command sets the first serial port on the first card for RS-232 communications as shown in the following example:

```
root@Moxa:~# setinterface
    Usage: setinterface device-node [interface-no]
    device-node - /dev/ttyM*
    interface-no    - following:
    none - to view now setting
    0 - set to RS232 interface
    1 - set to RS485-2WIRES interface
    2 - set to RS422 interface
    3 - set to RS485-4WIRES interface
root@Moxa:~#
```

Checking the Current Serial Settings

To check the current interface setting, type the following. The system should display a response as below.

```
root@Moxa: ~# setinterface /dev/ttyM0
Now setting is RS485-2WIRES interface.
```

In the example above, port 1 on card 1 is configured as a 2-wire RS-485 interface. After entering the lines of code below, port 1 gets reset as an RS-422 interface.

```
root@Moxa: ~# setinterface /dev/ttyM0 2
root@Moxa: ~# setinterface /dev/ttyM0
Now setting is RS422 interface.
```

ATTENTION

Serial interfaces ***will shift device node identifiers*** depending upon the location and number of cards mounted in the platform. E.g., if there are originally two cards mounted in the machine, and card number 1 is removed, then the second card's node identifier will change from **/dev/ttyM8** and **/dev/ttyM9** to **/dev/ttyM0** and **/dev/ttyM1**.
If you want to configure the machine for fixed serial interface device node identifiers, you can create a UDEV rule in **/etc/udev/rules.d/**. For help with this, consult the UDEV manual files, another Linux manual, or Moxa technical support for more details.

The system default for EPM-3032 interfaces is RS-232. By editing the device manager's rule scripts, it is possible to set all serial ports to one of the serial protocols (RS-485 or RS-422) instead. The steps below describe how to do so.

1. Edit Moxa's custom rules file for the device manager, which can be found at `/etc/udev/rules.d/96-moxa.rules`. Add the following command to `96-moxa.rules`:

```
# Set the device, EPM-3032, 0x1393:0x1022 default as 232 mode interface
DRIVERS=="mxser", ATTRS{vendor}=="0x1393", ATTRS{device}=="0x1022",
RUN+="/bin/setinterface /dev/ttyM%n 0"

"96-moxa.rules"
```

ATTENTION

The **VendorID** for the EPM-3032 is **0x1393m**, and the **DeviceID** is **0x1022**.

2. To change the default serial interface mode, edit the `setinterface` command you have just added to the Moxa rules file (`/etc/udev/rules.d/96-moxa.rules`). This will cause the Linux kernel to automatically set the V2400A module to your preferred interface at every reboot.
   - For RS-232, use `RUN+="/bin/setinterface /dev/ttyM%n 0"`
   - For RS-485 2-wire, use `RUN+="/bin/setinterface /dev/ttyM%n 1"`
   - For RS-422, use `RUN+="/bin/setinterface /dev/ttyM%n 2"`
   - For RS-485 4-wire, use `RUN+="/bin/setinterface /dev/ttyM%n 3"`
3. When finished, reboot your computer

```
Moxa:~# reboot
```

4. Once the computer restarts, confirm that the interfaces have been reset to the default settings.

```
Moxa:~# setinterface /dev/ttyM0
Now setting is RS485-2WIRES interface.
```

EPM-3438 Driver Installation

Upload the driver package to **/dev/shm**, a temporary file system on your computer.

```
root@Moxa:~# scp v2400a-epm3438_1.0.0_amd64.deb moxa@192.168.3.127:/dev/shm
```

Install the package:

```
root@Moxa:~# cd /dev/shm
root@Moxa: /dev/shm# dpkg -i v2400a-epm3438_1.0.0_amd64.deb
Selecting previously unselected package v2400a-epm3438.
(Reading database ... 48016 files and directories currently installed.)
Unpacking v2400a-epm3438 (from v2400a-epm3438_1.0.0_amd64.deb) ...
Setting up v2400a-epm3438 (1.0.0) ...
root@Moxa:/dev/shm#
```

After the driver is installed, you can use **lsmod** to check if the epm3438 module is loaded in the kernel.

```
Moxa:~# lsmod|more
Module                  Size  Used by
epm3438                12748  0
…
```

It will be loaded automatically when the system boots up.

To uninstall the driver, use the following command:

```
root@Moxa:~# dpkg --purge v2400a-epm3438
```

EPM-3438 Programming Guide

Digital I/O

Digital input/output channels are featured in some models of Moxa embedded computers, including the UC-7408, UC-8410, IA240, IA260, W406, and EPM-3438. These channels can be accessed at run-time for control or monitoring using the functions in the following sections. Each of the digital output (DO) channels can be individually set to high or low for each port, starting from 0. The digital input (DI) channels can be used to detect the state change of the digital input signal. The header file of digital I/O functions is **mxdgio.h**, which is located in the Linux **digit_input_change** directory.

Moxa functions for DI/DO

Function	**HANDLE mxdgio_epm3438_open(int HWIndex);**
Description	This function opens access to the DIO device.
Input	<HWIndex> The first or second EPM-3438 board.
Output	None
Return	When successful, this function gives access to the DIO device. Otherwise, there is an error.

Function	**void mxdgio_close(HANDLE fd);**
Description	This function closes the access to the DIO device.
Input	<fd> The access to the device.
Output	None
Return	None

Function	**int mxdgio_get_input_signal(HANDLE fd, int port);**
Description	This function gets the signal state of a digital input channel.
Input	<fd> The access to the device. <port> port #
Output	<state> DIO_HIGH (1) for high, DIO_LOW (0) for low
Return	Returns 1 for a high signal or 0 for a low signal, if successful. Otherwise, it returns a value of -1.

Function	**int mxdgio_get_output_signal(HANDLE fd, int port);**
Description	This function gets the signal state of a digital output channel.
Input	<fd> The access to the device. <port> Port number
Output	None
Return	Returns 1 for a high signal or 0 for a low signal, if successful. Otherwise, it returns a value of -1.

Function	**int mxdgio_set_output_signal_high(HANDLE fd, int port);**
Description	This function sets digital output channel to high.
Input	<fd> The access to the device. <port> Port number.
Output	none.
Return	When successful, this function returns 0. When an error occurs, it returns -1.

Function	**int mxdgio_set_output_signal_low(HANDLE fd, int port);**
Description	This function sets a digital output channel to low.
Input	<fd> The access to the device. <port> Port number.
Output	none.
Return	When successful, this function returns 0. When an error occurs, it returns -1.

Moxa I/O Control Definitions for COUNTER

This table shows the counter interface on the EMP-3438 module. If you want to read the counter value of the module, you can read it from COUNTER_NODE1. If you have the second EMP-3438 module, read it from COUNTER_NODE2.

```
#define    COUNTER_NODE1    "/dev/epm_3438_counter1"
#define    COUNTER_NODE2    "/dev/epm_3438_counter2"
```

Function	**mxdgio_epm3438_get_counter(int fd);**
Description	Gets the counter value
Input	<fd> The access to the counter device. <port> Port number.
Output	none.
Return	the counter value

Function	**mxdgio_epm3438_clear_counter(int fd);**
Description	Clears the counter value
Input	<fd> The access to the counter device. <port> Port number.
Output	none.
Return	0:clear success; -n: clear fail

mxdgio.h: Moxa Digital Input/Output Headers

1. The software CD included with your computer includes sample code to illustrate some implementations of common DI/DO functions. To find these sample files, navigate to the directory **/example/V2426A/EPM3438/digit_input_change** on the sample code CD that is bundled with your module; there, you will find the **mxdgio.h** file, which provides a convenient API for digital I/O and COUNTER programming. **Mxdgio.h** provides a set of macros and an API for programming either the digital I/O interfaces or the HARDWARE COUNTER.
2. The default initial value for digital output is **HIGH**. If you want to set the initial output status to **LOW**, you may instruct the kernel to load **epm_3438.ko** with a default **LOW** setting at boot time. To do this, edit the line epm3438 `epm3438_DO2LOW=1` in **/etc/modules**. To initialize the setting, reboot your computer.

```
Moxa: ~# vi /etc/modules


epm3438 epm3438_DO2LOW=1
```

Registering Digital I/O Callback Events

Moxa provides a library of functions that allow users to develop higher layer functions that respond to DI/O state changes. These functions allow user applications to create specific responses to digital I/O events by associating a callback function with an I/O event.

The source code files of the sample program are located in the `/example/V2100.V24XX/EPM3438/digit_input_change/` directory.
Four higher layer functions provide programmers with an API for timer callback events:

- `digit_io_timer_init`
- `digit_io_timer_dispatch`
- `digit_io_timer_add_callback`
- `digit_io_timer_dispatch_quit`

There are also four functions that give programmers an API for digital I/O callback events, available via the **digit_io_timer_add_callback** function:

- `DGTIO_GET_INPUT_STATE_CHANGE`
- `DGTIO_GET_INPUT`
- `DGTIO_GET_OUTPUT`
- `DGTIO_SET_OUTPUT`

The following example illustrates the use of the initialization function for registering a callback event:

```
mngr = digit_io_timer_init();
…
if (digit_io_timer_add_callback (mngr, HWIndex, port, DGTIO_GET_INPUT_STATE_CHANGE,
interval, input_chg_cb, &port) < 0) {
…
}
if (digit_io_timer_add_callback (mngr, HWIndex, port, DGTIO_GET_INPUT, interval,
input_get_cb, &port) < 0) {
…
}
if (digit_io_timer_add_callback (mngr, HWIndex, port, DGTIO_SET_OUTPUT, interval,
output_set_cb, &port) < 0) {
…
}
if (digit_io_timer_add_callback (mngr, HWIndex, port, DGTIO_GET_OUTPUT, interval,
output_get_cb, &port) < 0) {
…
}
digit_io_timer_dispatch(mngr);
```

Implementing Timer Functions on Digital IO Ports

The examples in this section show how to implement timer functions on Digital IO ports. The first example has two parts.

Example 1-1: (Routines to Operate Timer Function on a Digital IO Port.)

Folder and file: `/examples/ExpansionCard/LX/EPM3438/digit_input_change/digit_io_timer.c`

```
#include <stdio.h>
#include <stdlib.h>
#if !defined(_WIN32_WCE) && !defined(WIN32)
#include <time.h>
#endif
#include "digit_io_timer.h"
/* callback function */
static void
dgio_input_change_exec(DGIOMNGR *mngr, DGIOITEM *item)

{
   int sig;
   HANDLE fd=mngr->fd[item->HWIndex];
   switch(item->mode)

   {
   case DGTIO_GET_INPUT:
        sig = mxdgio_get_input_signal(fd, item->port);
        item->cb(item->HWIndex, item->port, sig, item->arg);
        break;
   case DGTIO_GET_OUTPUT:
        sig = mxdgio_get_output_signal(fd, item->port);
```

```
        item->cb(item->HWIndex, item->port, sig, item->arg);
        break;
    case DGTIO_GET_INPUT_STATE_CHANGE:
        sig = mxdgio_get_input_signal(fd, item->port);
        if (item->last_signal!=sig)
        {
            item->cb(item->HWIndex, item->port, sig, item->arg);
        }
        break;
    case DGTIO_SET_OUTPUT:
        sig = item->cb(item->HWIndex, item->port, item->last_signal, item->arg);
        if (sig)
        {
            mxdgio_set_output_signal_high(fd, item->port);
        }
        else
        {
            mxdgio_set_output_signal_low(fd, item->port);
        }
        break;
    default:
        return;
    }
    item->last_signal = sig;
}

/**** release the timer operation ****/
static void
dgio_input_change_release(DGIOMNGR *mngr)
{
    int i;
    DGIOITEM *item, *next;
     item=mngr->list;
     while(item)
     {
        next = item->next;
        free(item);
        item = next;
     }
     for ( i=0; i<HW_TOTAL; i++ )
        if (mngr->fd[i])
            mxdgio_close(mngr->fd[i]);
}

/****
This function initializes a timer manager
Returns: Return a pointer to the manager.
****/

DGIOMNGR*
digit_io_timer_init(void)
{
    DGIOMNGR *mngr;
    mngr = (DGIOMNGR*) calloc(1, sizeof(DGIOMNGR));
    if (mngr)
```

```
    {
    mngr->fd[0] = mxdgio_open();
#if 1 // Jared, 08-10-2010, support the second EPM-3438
    mngr->fd[1] = mxdgio_epm3438_open(0); // The first EPM-3438
    mngr->fd[2] = mxdgio_epm3438_open(1); // The second EPM-3438
#endif
        if (mngr->fd[0] < 0)
        {
            free(mngr);
            mngr = NULL;
        }
    }
    return mngr;
}

/***    adds a digital IO timer with a selected operation mode
    Inputs:    \
        <mngr> timer manager   \
        <HWIndex> specify which hardware device;   \
                    0: embedded DIO,    \
                    1: EPM-3438 #1,     \
                    2: EPM-3438 #2      \
        <port> specify which DIO pin      \
        <mode> the operation mode on the port      \
        <interval> the interval (in milliseconds) between 2 calls \
                    to a user-defined function      \
        <cb> the user-defined callback function    \
        <arg> argument to the function    \
    Returns:    \
        0 on sucess, otherwise failure    \
***/

int
digit_io_timer_add_callback(DGIOMNGR *mngr, int HWIndex, int port, int mode, int
interval, digit_io_cb_t cb, void *arg)
{
    DGIOITEM *item;
    item = (DGIOITEM*) calloc (1, sizeof (DGIOITEM));
    if (!item)
        return -1;
    item->next = mngr->list;
        mngr->list = item;
    item->cb  = cb;
    item->arg = arg;
    item->HWIndex = HWIndex; // Jared, 08-10-2010, HWIndex to support multiple boards
    item->port = port;
    item->mode = mode;
    item->interval = interval;
    item->next_time = interval;
    // Jared, 08-10-2010, HWIndex to support multiple boards
    item->last_signal =  mxdgio_get_input_signal(mngr->fd[HWIndex], port);
    return 0;
}
void
digit_io_timer_dispatch_quit(DGIOMNGR *mngr)
```

```
{
   if (mngr) mngr->dispatch = 0;
}

#define MAX_TIME 0XFFFFFFFF
/***  start and dispatch the timer operations   \
    Inputs:   \
        <mngr> the manager   \
    Returns:   \
        none   \
***/
void
digit_io_timer_dispatch(DGIOMNGR *mngr)
{
   DGIOITEM *item;
    unsigned int ms_sleep, n;
#if !defined(_WIN32_WCE) && !defined(WIN32)
   struct timeval to;
#endif
    mngr->dispatch = 1;
   while(mngr->list && mngr->dispatch)
   {
        for (item = mngr->list; item != NULL; item = item->next)
        {
            if (mngr->now_time < item->next_time)  /* not yet */
                continue;
        /*** overdue, executable ***/
            n = mngr->now_time - item->next_time;
        /*** move to the next time ***/
            item->next_time = mngr->now_time+item->interval-n;
            dgio_input_change_exec(mngr, item);
        }
        ms_sleep = MAX_TIME;
    /*** get the amount of time to sleep ***/
        for (item = mngr->list; item != NULL; item = item->next)
        {
            if (mngr->now_time < item->next_time)  /*** not yet ***/
            {
                n = item->next_time - mngr->now_time;
                if (n < ms_sleep) ms_sleep = n;
                continue;
            }
        }
        if (ms_sleep!=MAX_TIME)
        {
            #if !defined(_WIN32_WCE) && !defined(WIN32)
            to.tv_sec  = ms_sleep/1000;
            to.tv_usec = (ms_sleep%1000)*1000;
            if (select (0, NULL, NULL, 0, &to) != 0) /* sleep */
                break;
#else
            Sleep(ms_sleep);
#endif
            mngr->now_time += ms_sleep;
        }
```

```
    }
    dgio_input_change_release(mngr);
}
```

Example 1-2: (Main Routines and Callbacks to Operate Timer Functions on Digital IO Ports.)

Folder and file: /examples/ExpansionCard/LX/EPM3438/digit_input_change/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "digit_io_timer.h"
static int
input_chg_cb(int HWIndex, int port, int sig, void *arg)
{
       printf("input_chg_cb() HWIndex %d port %d sig %d\n", HWIndex, port, sig);
       return 0;
}
static int
input_get_cb(int HWIndex, int port, int sig, void *arg)
{
       printf("input_get_cb() HWIndex %d port %d sig %d\n", HWIndex, port, sig);
       return 0;
}
static int
output_set_cb(int HWIndex, int port, int last_sig, void *arg)
{
       printf("output_set_cb() HWIndex %d port %d last sig %d\n", HWIndex, port,
last_sig);
       last_sig++;
       last_sig %= 2;
       printf("new sig=%d\n", last_sig);
       return last_sig;
}
static int
output_get_cb(int HWIndex, int port, int sig, void *arg)
{
       printf("output_get_cb() HWIndex %d port %d sig %d\n", HWIndex, port, sig);
       return 0;
}
#define INTERVAL      10000
int
#if defined(_WIN32_WCE)
WINAPI
WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow )
#else
main(int argc, char *argv[])
#endif
{
       DGIOMNGR *mngr;
       int HWIndex;
       int port;
       int interval;
```

```
#if defined(_WIN32_WCE)
	int    argc;
	char cmdline[256], *argv[32];
	WideCharToMultiByte(CP_ACP, 0, (LPCTSTR)lpCmdLine, 255, cmdline, 256, NULL,
NULL);
	argc = split_line(argv+1, 32, cmdline)+1;
#endif
	if (argc > 1) interval = atoi(argv[1]);
	else interval = INTERVAL;
	mngr = digit_io_timer_init();
	if (mngr == NULL) {
		printf("digit_io_timer_init() error\n");
		return -1;
	}
	HWIndex=0; // HWIndex=0 for embedded DIO
	for (port = 0; port < 1; port++) {
	   if (digit_io_timer_add_callback(mngr, HWIndex, port,
DGTIO_GET_INPUT_STATE_CHANGE, interval, input_chg_cb, &port) < 0) {
			printf("add %d input change callback error\n", port);
			    return -2;
	      }
	      if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_GET_INPUT,
interval, input_get_cb, &port) < 0) {
	         printf("add %d input callback error\n", port);
	         return -3;
	      }
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_SET_OUTPUT, interval,
output_set_cb, &port) < 0) {
	         printf("add %d set output callback error\n", port);
	         return -4;
	      }
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_GET_OUTPUT, interval,
output_get_cb, &port) < 0) {
	         printf("add %d get output callback error\n", port);
	         return -5;
	      }
	}
	/ / HWIndex=1 for EPM-3438 board #1; HWIndex=2, for EPM-3438 board #2
	for (HWIndex = 0; HWIndex < HW_TOTAL; HWIndex++ ) {
		for (port = 0; port < 8; port++) {
		      /* since list is LIFO last callbacks are added first */
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_GET_INPUT_STATE_CHANGE,
interval, input_chg_cb, &port) < 0) {
				printf("add %d input change callback error\n", port);
				return -2;
			}
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_GET_INPUT, interval,
input_get_cb, &port) < 0) {
				printf("add %d input callback error\n", port);
				return -3;
			}
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_SET_OUTPUT, interval,
output_set_cb, &port) < 0) {
				printf("add %d set output callback error\n", port);
				return -4;
```

```
                    }
if (digit_io_timer_add_callback(mngr, HWIndex, port, DGTIO_GET_OUTPUT, interval,
output_get_cb, &port) < 0) {
                            printf("add %d get output callback error\n", port);
                            return -5;
                    }
              }
       }
       digit_io_timer_dispatch(mngr);
       return 0;
}
```

Example 2: (Reading the EPM-3438 Counter Value and Clearing the Counter)

Folder and file: /examples/ExpansionCard/LX/EPM3438/digit_input_change/tcounter.c

```
#include      <stdio.h>
#include      <stdlib.h>
#include      <sys/time.h>
#include      <fcntl.h>
#include      <unistd.h>
#include      <signal.h>
#include      "mxdgio.h"  // For counter reading or clear
#define COUNTER_NODE1   "/dev/epm_3438_counter1" // The first EPM-3438
#define COUNTER_NODE2   "/dev/epm_3438_counter2" // The second EPM-3438
int main(int argc, char * argv[])
{
       int retval;
       int fd, fd2, len;
       unsigned int counter_value;
       fd=open(COUNTER_NODE1, O_RDONLY);
       while( 1 ) {
              printf("\nSelect a number of menu, other key to exit.   \n\
       1. Get counter value                        \n\
       2. Clear the counter                         \n\
       Others. quit                               \n\
Choose : ");
              scanf("%d", &retval);
              if ( retval == 1 ) {    // Get counter without reset
                     counter_value = mxdgio_epm3438_get_counter(fd);
                     printf("EPM-3438 board #1 counter:%d\n", counter_value);
              }
              else if ( retval == 2 ) { // Get counter with reset
                     retval = mxdgio_epm3438_clear_counter(fd);
                     if ( retval < 0 )
                            printf("EPM-3438 board #1 counter reset fail\n");
              }
              else {
                     break;
              }
       }
       close(fd);
       return 0;
```

```
}
```

EPM-3112 CAN Bus Interface

The EPM-3112 module provides V2400A series computers with a CAN bus interface. CAN is a broadcast serial bus standard for connecting electronic control units (ECUs). Each node is able to send and receive messages, but not simultaneously: a message (consisting primarily of an ID—usually chosen to identify the message-type/sender—and up to eight message bytes) is transmitted serially onto the bus, one bit after another. This signal-pattern codes the message (in NRZ) and is sensed by all nodes.

The Moxa EPM-3112 module provides the CAN bus interface for industrial CAN communication. Users can use Moxa's CAN library or the local GNU/Linux device control interface (ioctl) to program reads, writes, and controls for CAN devices.

EPM-3112 Driver Installation

CAN is a serial bus protocol for connecting and controlling electronic devices in harsh environments. A CAN bus connects ECUs (electronic control units) so that each node may send and receive messages that consist of an ID (to identify the message-type and sender) and up to eight message bytes.

The Moxa EPM-3112 module provides the CAN bus interface for industrial CAN communication. The V2400A series computer provides the SocketCAN interface for industrial CAN communication. The SocketCAN concept extends the Berkeley sockets API in Linux by introducing a new protocol family, PF_CAN that coexists with other protocol families like PF_INET for the Internet Protocol.

Users can use the file control interface to read, write, or control the CAN interface as a file for easy CAN programming.

To install the EPM-3112 kernel module:

1. Use the **dpkg** installer to install the **v2400a-epm3112_1.0.0_amd64.deb** package. This package will automatically enable your EPM-3112 module to load at boot time:
   ```
   Moxa: ~# dpkg -i v2400a-epm3112_1.0.0_amd64.deb
   ```
2. After installation, please reboot your device.

EPM-3112 Configuring the Socket CAN Interface

After the modules are loaded, use the **ip link** command to check the CAN device.

```
Moxa:~# ip link
can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10
   link/can
can1: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10
   link/can
```

The next step is to configure the CAN interface and start it as a standard net interface. Here's an example with bitrate 12500:

```
# ip link set can0 up type can bitrate 12500
# ip link set can1 up type can bitrate 12500
```

After using the SocketCAN API, the SocketCAN information is located in /proc/net/can; use the following command to determine the version:

```
# cat /proc/net/can/version
```

Use the following command to get the statistics:

```
# cat /proc/net/can/stats
```

EPM-3112 Programming Guide

The following code is a working example of the SocketCAN API, which sends packets using the raw interface. It is based on the notes documented in the Linux Kernel (https://www.kernel.org/doc/Documentation/networking/can.txt).

Folder and file: /examples/ExpansionCard/LX/EPM3112/can_read_write/**can_write.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <linux/can.h>
#include <linux/can/raw.h>

int
main(void)
{
    int s;
    int nbytes;
    struct sockaddr_can addr;
    struct can_frame frame;
    struct ifreq ifr;

    char *ifname = "can1";

    if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Error while opening socket");
        return -1;
    }

    strcpy(ifr.ifr_name, ifname);
    ioctl(s, SIOCGIFINDEX, &ifr);

    addr.can_family  = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;

    printf("%s at index %d\n", ifname, ifr.ifr_ifindex);

    if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("Error in socket bind");
        return -2;
    }

    frame.can_id  = 0x123;
    frame.can_dlc = 2;
    frame.data[0] = 0x11;
    frame.data[1] = 0x22;
```

```
    nbytes = write(s, &frame, sizeof(struct can_frame));

    printf("Wrote %d bytes\n", nbytes);

    return 0;
}
```

The following sample code illustrates how to read the data.

File and Folder: /examples/ExpansionCard/LX/EPM3112/can_read_write/**can_read.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <linux/can.h>
#include <linux/can/raw.h>

int
main(void)
{
	int i;
	int s;
	int nbytes;
	struct sockaddr_can addr;
	struct can_frame frame;
	struct ifreq ifr;

	char *ifname = "can0";

	if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
		perror("Error while opening socket");
		return -1;
	}

	strcpy(ifr.ifr_name, ifname);
	ioctl(s, SIOCGIFINDEX, &ifr);

	addr.can_family  = AF_CAN;
	addr.can_ifindex = ifr.ifr_ifindex;

	printf("%s at index %d\n", ifname, ifr.ifr_ifindex);

	if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
		perror("Error in socket bind");
		return -2;
	}

	nbytes = read(s, &frame, sizeof(struct can_frame));
```

```
        if (nbytes < 0) {
                perror("Error in can raw socket read");
                return 1;
        }

        if (nbytes < sizeof(struct can_frame)) {
                fprintf(stderr, "read: incomplete CAN frame\n");
                return 1;
        }

        printf("  %5s  %03x   [%d] ", ifname, frame.can_id, frame.can_dlc);
        for (i = 0; i < frame.can_dlc; i++)
                printf(" %02x", frame.data[i]);
        printf("\n");

        return 0;
}
```

Installing the EPM-DK02 Kernel Module

Moxa's EPM-DK02 module supports 2 mini PCIe sockets for connecting mini PCIe modules.

- **Socket 1, Physical interface:** mini PCIe
  Electrical interface: mini PCIe, USB 2.0.
- **Socket 2, Physical interface:** mini PCIe
  Electrical interface: USB 2.0.

EPM-DK02 Kernel Module

Upload the v2400a-dkcontrol_1.0.0_amd64.deb file to the target machine and use **dpkg** command to install the package.

```
Moxa:/home# dpkg -i v2400a-dkcontrol_1.0.0_amd64.deb
```

Configuring Power Control

The EPM-DK02 and EPM-DK03 modules come with the capability to automate a modular device's power status. Note, however, that this function is provided primarily for powering on and off GPRS/HSDPA cards that use a USB interface. It is **NOT** advisable to use this function with PCIe devices, because doing so may damage them.

WARNING

Using the EPM-DK02's or EPM-DK03's onboard power controls to control PCIe or USB-DOM devices is not recommended. Using the onboard power controls for control of PCIe and USB-DOM devices may damage the devices (PCIe) or corrupt data (USB-DOM). ***Any use of the EPM-DK02's automated, onboard power controls with PCIe and USB-DOM devices is undertaken at the user's own risk.***

It is also not advisable to use the power control feature with USB DOM devices, because these devices involve the mounting and unmounting of file systems, and while the Linux system will be able to automatically mount the USB DOM file systems as they come online, without careful scripting it will not be able to automatically unmount the file systems once the device goes offline.

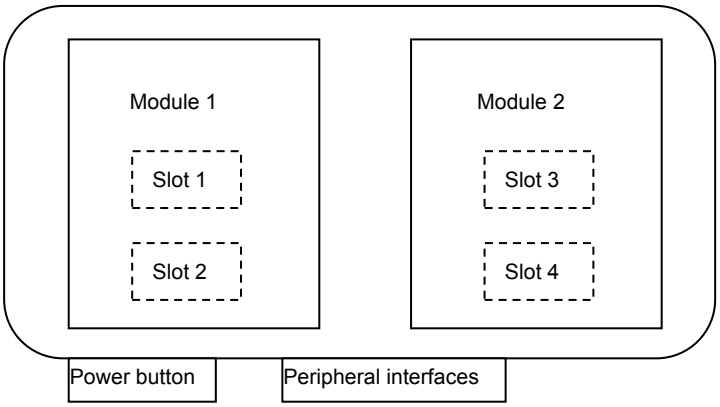
The command for manipulating the PCIe card's power feature is the **mx-dkcontrol**. The precise syntax is as follows, with **slot_number** indicating the number of the card slot located on the EPM-DK02 or EPM-DK03 board itself (note: slot_number does not refer to module slots on the V2400A computer itself):

```
Moxa:~# mx-dkcontrol
Usage: mx-dkcontrol [slot_num] [1/0]
       [slot_num] :
              1. slot1 socket1 PCIe/USB
              2. slot1 socket2 USB
              3. slot2 socket1 PCIe/USB
              4. slot2 socket2 USB
       [1/0] :
              0. Power off
              1. Power on
```

Fox example, if you want to power off socket1 PCIe/USB connectivity on slot 1, issue the following command:

```
Moxa:~# mx-dkcontrol 1 0
```

Note that the slot number will depend on the position of the interface within the card:



ATTENTION

The major aim of the **mx-dkcontrol** command is ***to reset a USB GPRS/HSPDA module***; it is not advisable to use the power on-off feature with a USB DOM module. This is because when a USB DOM module is powered on, the Linux system will automatically mount its partitions, but when it is powered off the system cannot automatically unmount its partitions.

Any use of the EPM-DK02 or EPM-DK03's automated, onboard power controls with PCIe interfaces and USB-DOM devices is undertaken at the user's own risk.

WARNING

Note that the power control is only suitable for devices that have a USB interface. If you are using a device with a PCIe interface, do not enable the power on/off control function, since doing so could damage the device.

Any use of the EPM-DK02 or EPM-DK03's automated, onboard power controls with PCIe interfaces and USB-DOM devices is undertaken at the user's own risk.

Wi-Fi Module

In this section we show you how to connect to an 802.11 access point. The connection program we will use is **wpa_supplicant**.

There are two ways to use **wpa_supplicant**. You can use **wifi_mgmt**, which is offered by Moxa or use the use `wpa_supplicant` command.

Install wifi_mgmt

Upload the v2400a-wifimgmt_1.0.0_amd64.deb file to target machine and use dpkg installer to install the package.

```
Moxa:/home# dpkg -i v2400a-wifimgmt_1.0.0_amd64.deb
```

wifi_mgmt Usage

Manual page

wifi_mgmt help

The **wifi_mgmt** utility is used for handling Wi-Fi module related behavior.

```
moxa@Moxa:~$ sudo wifi_mgmt help
[sudo] password for moxa:
Usage:
     /sbin/wifi_mgmt [OPTIONS]

OPTIONS
      start Type=[type] SSID=[ssid] Password=[password]
            Insert an AP information to the managed AP list and then connect to the
AP.

            [type]       open/wep/wpa/wpa2
            [ssid]       access point's SSID
            [password]    access point's password

            example:
               wifi_mgmt start Type=wpa SSID=moxa_ap Password=moxa
               wifi_mgmt start Type=open SSID=moxa_ap
      start [num]
            Connect to AP by the managed AP list number.
      start
            Connect to the last time AP that was used.
      scan -d
            Scan all the access points information and show the detail message.
      scan
            Scan all the access points information.
      signal
            Show the AP's signal.
      list
            Show the managed AP list.
      insert Type=[type] SSID=[ssid] Password=[password]
            Insert a new AP information to the managed AP list.

            [type]       open/wep/wpa/wpa2
            [ssid]       access point's SSID
            [password]    access point's password

            example:
               wifi_mgmt insert Type=wpa SSID=moxa_ap Password=moxa
      select [num]
            Select an AP num to connect which is in the managed AP list.
```

```
      stop
            Stop network.
      status
            Query network connection status.
      interface [num]
            Switch to another wlan[num] interface.

            [num]    interface number
            example:
               wifi_mgmt interface 0
      interface
            Get the current setting interface.
      reconnect
            Reconnect to the access point.
      restart
            Stop wpa_supplicant then start it again.
      version
            Wifi management version.
```

Connecting to an AP

There are three ways to connect to an AP. The DNS and default gateway will be configured automatically. If you want to use the wireless interface's gateway, be sure to clean up your computer's default gateway first.

wifi_mgmt start Type=[type] SSID=[ssid] Password=[password]

Insert the AP information in the managed AP list and then connect to an AP.

```
root@Moxa:~# wifi_mgmt start Type=wpa SSID=moxa_ap Password=moxa
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

wifi_mgmt start [num]

Connect to the AP using the managed AP list number. If you have inserted AP information before, some AP information will still be in the managed AP list. Check the managed AP list with the wifi_mgmt list command.

```
root@Moxa:~# wifi_mgmt list
network id / ssid / bssid / flags
0    MOXA_AP1  any    [LAST USED]
1    MOXA_AP2  any    [DISABLED]
2    MOXA_AP3  any    [DISABLED]
```

Choose an AP number to start.

```
root@Moxa:~# wifi_mgmt start 1
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

wifi_mgmt start

Connect to the previous AP that was used.

```
root@Moxa:~# wifi_mgmt list
network id / ssid / bssid / flags
0    MOXA_AP1  any    [LAST USED]
1    MOXA_AP2  any    [DISABLED]
2    MOXA_AP3  any    [DISABLED]
```

Use the command wifi_mgmt to connect to the AP "MOXA_AP1" that was used the previous time.

```
root@Moxa:~# wifi_mgmt start
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

Stop or restart network

wifi_mgmt stop

```
root@Moxa:~# wifi_mgmt stop
wpa_supplicant is closed!!
```

wifi_mgmt restart

```
root@Moxa:~# wifi_mgmt restart
wpa_supplicant is closed!!
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

Insert an AP or choose another AP to connect.

If you want to use another AP to connect, use the wifi_mgmt select command to switch to another AP.

```
root@Moxa:~# wifi_mgmt insert Type=wpa2 SSID=MOXA_AP3 Password=moxa
root@Moxa:~# wifi_mgmt list
network id / ssid / bssid / flags
0    MOXA_AP1  any    [CURRENT]
1    MOXA_AP2  any    [DISABLED]
2    MOXA_AP3  any    [DISABLED]
```

If you want to use another AP to connect, use the wifi_mgmt select command to switch to another AP.

```
root@Moxa:~# wifi_mgmt list
network id / ssid / bssid / flags
0    MOXA_AP1  any    [DISABLED]
1    MOXA_AP2  any    [CURRENT]
2    MOXA_AP3  any    [DISABLED]
root@Moxa:~# wifi_mgmt select 2
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

Other functions

wifi_mgmt scan

Scan all of the access point information.

```
root@Moxa:~# wifi_mgmt scan
bssid / frequency / signal level / flags / ssid
b0:b2:dc:dd:c9:e4      2462    -57    [WPA-PSK-TKIP][ESS]    WES_AP
fc:f5:28:cb:8c:23      2412    -57    [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fe:f0:28:cb:8c:23      2412    -59    [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fc:f5:28:cb:39:08      2437    -79    [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fe:f0:28:cb:39:08      2437    -81    [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fc:f5:28:cb:5d:a8      2462    -83    [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
```

```
2c:54:cf:fd:5a:cf      2437    -83     [WPA-PSK-TKIP][ESS]     5566fans
fe:f0:28:cb:5d:a8      2462    -87     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fe:f0:28:cb:5d:78      2462    -89     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fe:f0:28:cb:39:11      2437    -89     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fc:f5:28:cb:39:11      2437    -91     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fe:f0:28:cb:39:0b      2412    -91     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
02:1a:11:f1:dc:a1      2462    -91     [WPA2-PSK-CCMP][ESS]    M9 Davidoff
fc:f5:28:cb:5d:78      2462    -93     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fe:f0:28:cb:5d:b7      2462    -93     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-Mobile
fc:f5:28:cb:39:0b      2412    -93     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fc:f5:28:cb:5d:b7      2462    -95     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
fc:f5:28:cb:5d:93      2462    -97     [WPA2-EAP-CCMP-preauth][ESS]    MHQ-NB
```

wifi_mgmt scan -d

Scan all of the access point information and show a detailed message.

```
root@Moxa:~# wifi_mgmt scan -d
wlan0    Scan completed :
        Cell 01 - Address: FC:F5:28:CB:8C:23
                Channel:1
                Frequency:2.412 GHz (Channel 1)
                Quality=51/70  Signal level=-59 dBm
                Encryption key:on
                ESSID:"MHQ-NB"
                        9 Mb/s; 12 Mb/s; 18 Mb/s
                Mode:Master
                   Group Cipher : CCMP
                   Pairwise Ciphers (1) : CCMP
                   Authentication Suites (1) : 802.1x
                  Preauthentication Supported
        Cell 02 - Address: FE:F0:28:CB:5D:A8
                Channel:11
                Frequency:2.462 GHz (Channel 11)
                Quality=25/70  Signal level=-85 dBm
                Encryption key:on
                ESSID:"MHQ-Mobile"
                        9 Mb/s; 12 Mb/s; 18 Mb/s
                Mode:Master
                   Group Cipher : CCMP
                   Pairwise Ciphers (1) : CCMP
                   Authentication Suites (1) : 802.1x
                  Preauthentication Supported
More.. .. ..
```

wifi_mgmtsignal

Shows the AP's signal level.

```
root@Moxa:~# wifi_mgmt signal
level=-59 dBm
```

wifi_mgmt delete

```
root@Moxa:~# wifi_mgmt list
network id / ssid / bssid / flags
0    MOXA_AP1  any    [CURRENT]
1    MOXA_AP1  any    [DISABLED]
2    MOXA_AP3  any    [DISABLED]
```

```
root@Moxa:~# wifi_mgmt delete 2
***** WARNING *****
Are you sure that you want to delete network id 2 (y/n)y
network id / ssid / bssid / flags
0    MOXA_AP1  any
1    MOXA_AP2  any    [DISABLED]
```

wifi_mgmt status

```
root@Moxa:~# wifi_mgmt status
bssid=b0:b2:dc:dd:c9:e4
ssid=MOXA_AP1
id=0
mode=station
pairwise_cipher=TKIP
group_cipher=TKIP
key_mgmt=WPA-PSK
wpa_state=COMPLETED
ip_address=192.168.1.36
address=00:0e:8e:4c:13:5e
```

wifi_mgmt interface [num]

If there is more than one Wi-Fi interface, you can change the interface.

```
root@Moxa:~# wifi_mgmt interface
There is(are) 2 interface(s):
wlan0   [Current]
wlan1
root@Moxa:~# wifi_mgmt interface 1
Now is setting the interface as wlan1.
```

wifi_mgmt reconnect

```
root@Moxa:~# wifi_mgmt reconnect
wpa_state=SCANNING
wpa_state=SCANNING
wpa_state=COMPLETED
*** Get DHCP IP address from AP ***
*** Get DHCP IP from AP! ***
```

wifi_mgmt version

```
root@Moxa:~# wifi_mgmt version
wifi_mgmt version 1.0 Build 15050223
```

Configuring the Wireless LAN Using wpa_supplicant.conf

WARNING

You might encounter **compatibility issues** if you configure Wi-Fi settings using **wifi_mgmt** instead of editing the `wpa_supplicant.conf` file. The **`wifi_mgmt`** command edits the wpa_supplicant.conf file dynamically. Hence, it is convenient to use the **`wifi_mgmt`** command to configure wireless LAN instead of editing the wpa_supplicant.conf file.

Moxa strongly advises against using the WEP and WPA encryption standards. Both are now officially deprecated by the Wi-Fi Alliance, and are considered insecure. To guarantee proper Wi-Fi encryption and security, please use WPA2 with the AES encryption algorithm.

You can configure the Wi-Fi connection using a configuration file or the **`wpa_supplicant`** command.

The following example is for OPEN/WEP/WPA/WPA2 AP.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel

update_config=1

### Open system ###
#network={
#       ssid="Open"
#       key_mgmt=NONE
#}
###################

##### WEP #####
#network={
#       ssid="WEP-ssid"
#       bssid=XX:XX:XX:XX:XX:XX
#       key_mgmt=NONE
#       wep_key0=KEY
#}
###############

##### WPA/WPA2 PSK #####
#network={
#       ssid="WPA-ssid"
#       proto=WPA WPA2 RSN
#       key_mgmt=WPA-PSK
#       pairwise=TKIP CCMP
#       group=TKIP CCMP
#       psk="KEY"
#}
######################
```

The basic command to connect for WPA-supplicant is:

root@Moxa:~# wpa_supplicant -i <interface> -c <configuration file> -B

The **-B** option should be included because it forces the supplicant to run in the background.

1. Connect with the following command after editing the wpa_supplicant.conf file:
   root@Moxa:~# wpa_supplicant -i wlan0 -c /etc/wpa_supplicant.conf -B
2. Use **iwconfig** to check the connection status. The response you receive should be similar to the following:

```
wlan0     IEEE 802.11abgn  ESSID:"MOXA_AP"
          Mode:Managed  Frequency:2.462 GHz  Access Point: 00:1F:1F:8C:0F:64
          Bit Rate=36 Mb/s   Tx-Power=27 dBm
          Retry min limit:7   RTS thr:off   Fragment thr:off
          Encryption key:1234-5678-90   Security mode:open
          Power Management:off
          Link Quality=37/70  Signal level=-73 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

ATTENTION

For more information about wpa_supplicant.conf, go to the following websites:
http://www.daemon-systems.org/man/wpa_supplicant.conf.5.html
http://linux.die.net/man/5/wpa_supplicant.conf

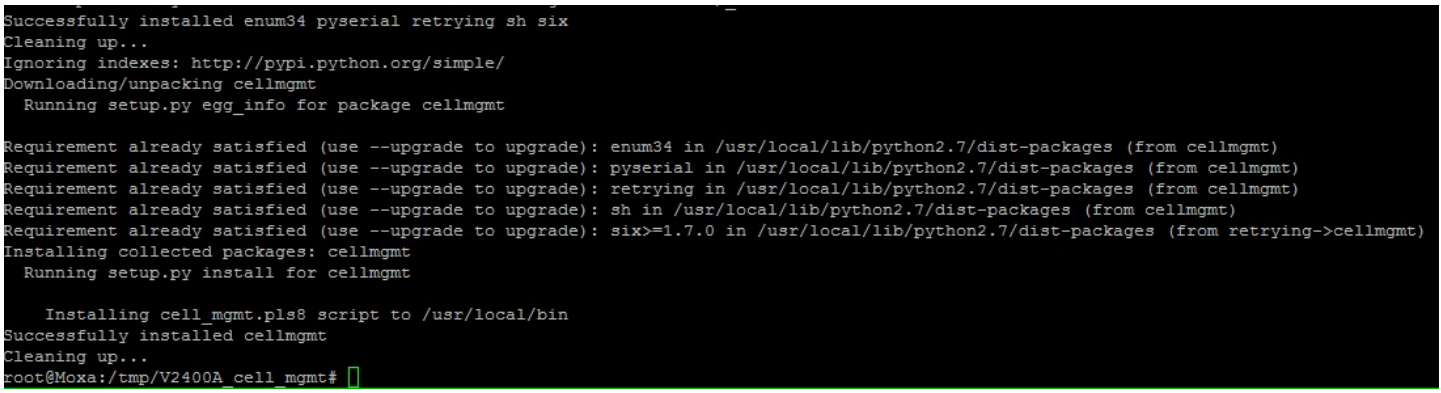
Cellular Module (PLS8-X R3 and Above)

The DK02/DK03 cards in the V2400A expansion modules are provided with two mini PCIe sockets in which cellular modules can be installed. You can use the PLS8-X cellular modules with a V2400A-LX series expansion module. Specifications of the cellular modules that you can use with the V2400A are available in the product datasheet. Contact your sales representative for more information.

Once the cellular module is connected, use the V2400A-LX cellular connection utility **`cell_mgmt`** to establish a cellular connection. Commands to determine the signal strength, use the dial-up function on the V2400A-LX, and other advanced functions are explained in the sections below:

Installing cell_mgmt

Copy the **`utility\LX\V2400A-pls8_cell_mgmt`** directory from the CD to the V2400A machine and install it.



Using cell_mgmt

Usage Menu

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt
ERROR: no argument given
Usage:
        start APN=<apn> [PIN=<pin>]
                Start network.

                example:
                        cell_mgmt.pls8 start
                        cell_mgmt.pls8 start APN=internet
                        cell_mgmt.pls8 start APN=internet PIN=0000
        stop
                Stop network.
        status
                Query network connection status.
        signal
                Get signal strength.
        sim_status
                Query sim card status.
        set_pin <pin>
                Set PIN code to configuration file.
        m_info
                Module information.
        operator
                Telecommunication operator.
        at <'AT command'>
                Input AT Command.
                Must use SINGLE QUOTATION to enclose AT Command.
        version
                Cellular management version.

root@Moxa:/tmp/V2400A_cell_mgmt#
```

start

Automatically sets the DNS and default gateway for the cellular interface and establishes a connection.

Before you set a gateway for the cellular interface on your computer make sure you disable the computer's default gateway.

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt start APN=internet PIN=0000
Connected
```

stop

Disconnects the cellular network connection.

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt stop
Disconnected
```

status

Gives the status of the network connection [connected, disconnected]

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt status
connected
```

signal

Gives the signal strength of the cellular module.

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt signal
-77 dbm
```

sim_status

Gives the SIM card status [READY, LOCKED, FAILED]

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt sim_status
READY
```

m_info

Shows information on the cellular module.

```
root@Moxa:/tmp/V2400A_cell_mgmt# cell_mgmt m_info
Module=Cinterion PLS8
WWAN_node=usb0
AT_port=/dev/ttyACM0
GPS_port=
LAC=083F
CellId=326D
ICC-ID=89886920032016961209
IMEI=004401081424513
QMI_port=
root@Moxa:/tmp/V2400A_cell_mgmt#
```

Installing the EPM-DK03 Module

The EPM-DK03 provides a combination GPS/GPRS card with an extra card slot where a second Wi-Fi, GPS, or GPRS card may be installed. The kernel module for the GPS card bundled with the EPM-DK03 is precompiled into the Linux kernel binary, so there is no need to install another one. The GPS card may be accessed and controlled using the GPS daemon (GPSd), over the ports **/dev/ttyACM0**, or **/dev/ttyACM1** (if a 2nd GPS module is installed). To set up the kernel module:

1. First check if the GPS card is transmitting raw data by issuing the following command to the device node, /dev/ttyACM0. If no data is being returned by the card, first try adjusting the GPS antenna to troubleshoot the problem. If there is no way of establishing reception, contact Moxa technical support at the phone number provided in the title plate of this manual.

```
Moxa:~# cat /dev/ttyACM0
$GPGSV,1,1,04,24,28,123,37,21,09,054,31,19,52,213,,23,47,270,*74
$GPGGA,061824.0,2458.835139,N,12133.055835,E,1,05,19.7,-103.5,M,,,,*14
$GPRMC,061824.0,A,2458.835139,N,12133.055835,E,,,290710,,,A*68
$GPGSA,A,3,24,21,06,31,16,,,,,,,,25.5,19.7,18.5*29
$GPVTG,,T,,M,0.0,N,0.0,K*4E
```

2. Next, install the Linux GPS daemon from public repositories. GPSd is the GPS background interface that will communicate with the raw GPS device. First, terminate the cat process you have just initiated using
   `Moxa:~# killall cat`
   and then install the GPS daemon as follows:

```
Moxa:~# apt-get install gpsd
```

ATTENTION

If you do not wish to expose the computer to the open Internet, then you may prepare a software CD in advance and use that, instead. This will require an alteration of your `apt.source` list, however. For more information on how to do this, consult this web page:

http://answers.oreilly.com/topic/19-how-to-install-debian-packages-from-cd-rom/

3. Start the GPS daemon:

```
Moxa:~# /etc/init.d/gpsd start
```

Installing the GPS Test Clients

Next, you must install test clients for gpsd. **Xgps** is a simple X interface test client that displays GPS position, time, and velocity information along with the location of accessible satellites. **Cgps** is similar to xgps, but is able to run over a serial or terminal interface and does not feature the pictorial satellite display.

1. You may install **cgps** and **xgps** using Debian's public repositories, accessible over the Internet:

```
Moxa:~# apt-get install gpsd-clients
```

ATTENTION

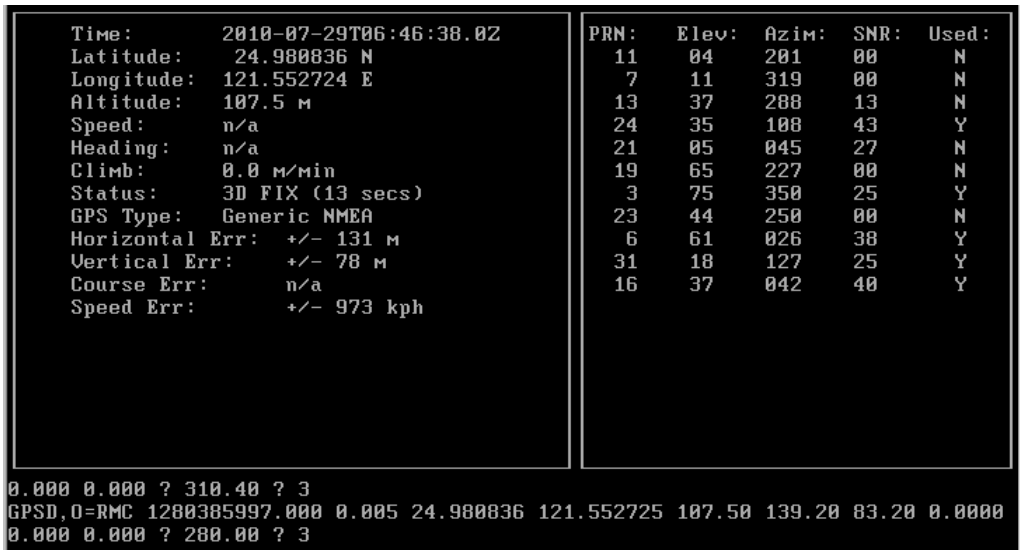
If you do not wish to expose the computer to the open Internet, then you may prepare a software CD in advance and use that, instead. This will require an alteration of your apt.source list, however. For more information on how to do this, consult this web page:

http://answers.oreilly.com/topic/19-how-to-install-debian-packages-from-cd-rom/

2. You may now use **cgps** and **xgps** to query the GPS daemon. **Xgps** is used on the desktop, and **cgps** is used on the command line terminal, or over a serial emulator/interface. You may access either client by logging in remotely, using SSH or a virtual desktop. To get a basic report on the current GPS data, call **cgps** on the console:

   `Moxa~#: cgps`

   and you should a report that looks something like this:



NOTE You may call the Unix man pages for more information on configuring and using the GPS daemon (man gpsd) or the GPS client (man cgps). Or visit the GPSd project website for more completely documentation: http://gpsd.berlios.de/

Windows System

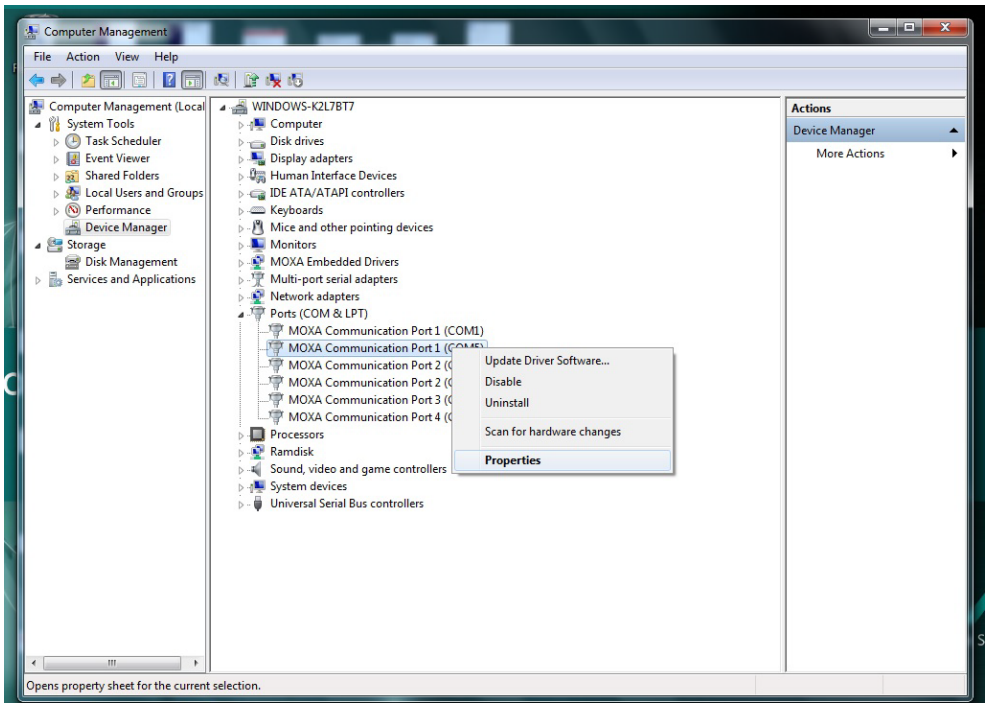
EPM-3032: Driver Installation

The EPM-3032 driver has existed in V2426A originally.

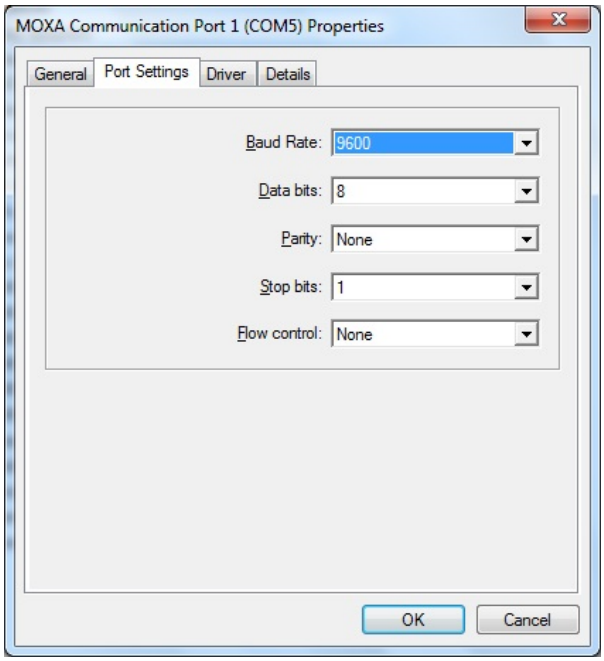
EPM-3032: Configuring Serial Port Mode

Take the following steps to configure the operation mode of each COM port:

1. Go to the **Control Panel Ports (COM & LPT)** and select the COM port. For example, MOXA Port 1 (COM5).
2. Right-click the COM port and then click **Properties**.



3. Click on the **Port Settings** tab and specify the values of the COM port settings.

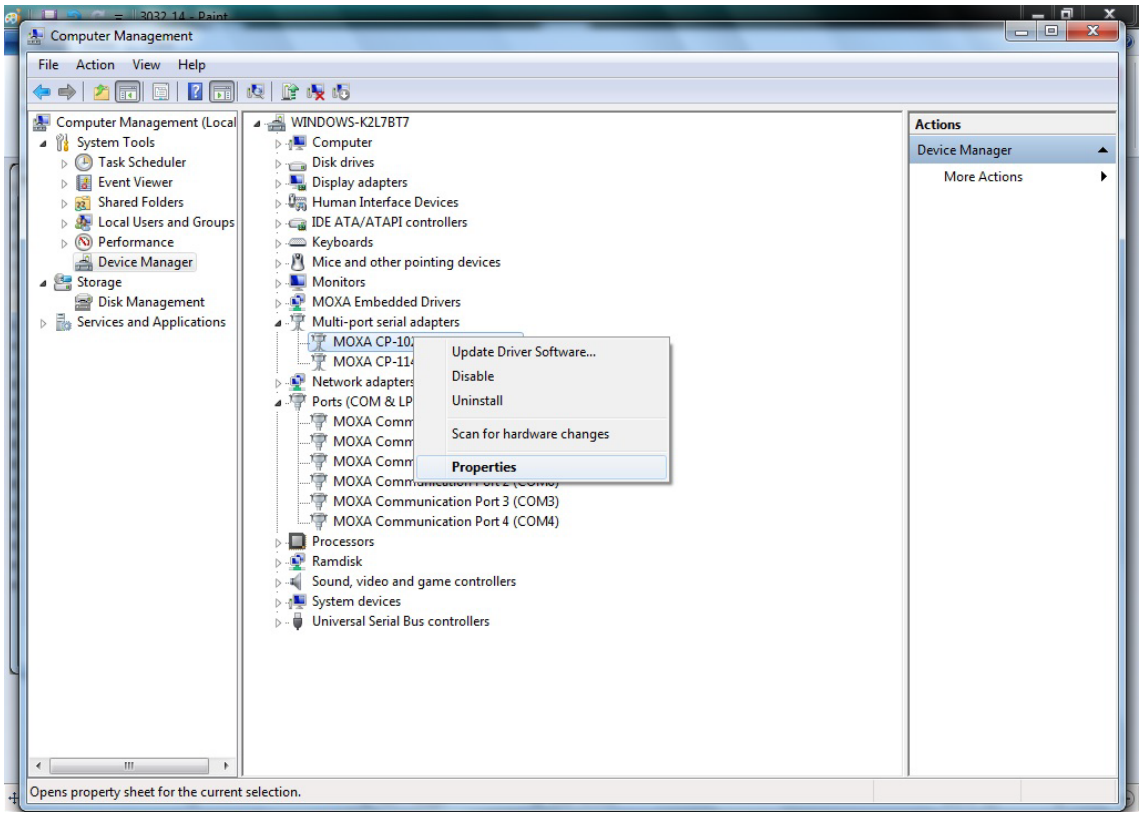


4. Click **OK** to apply the settings.

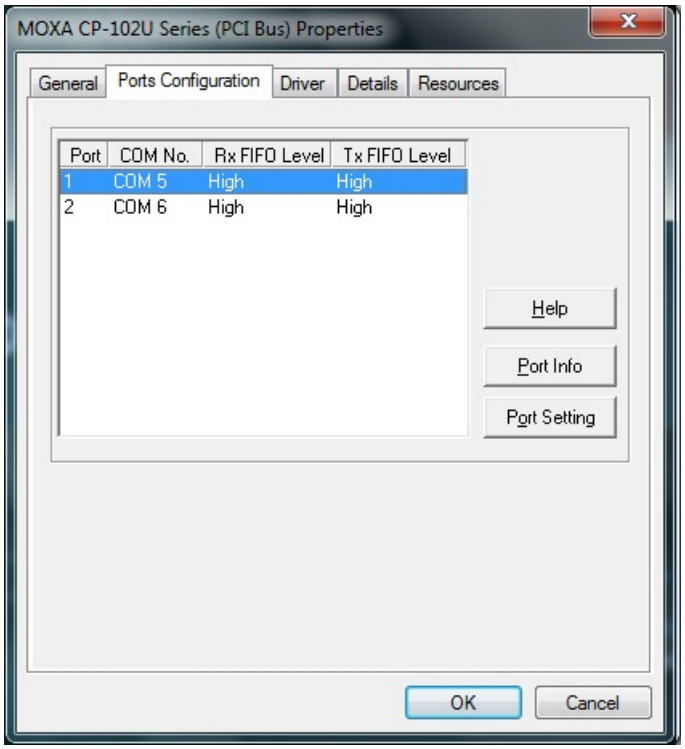
In some situations, you might want to change the COM port name to match the name used by your program. Take the following steps to change the COM port names:

1. Go to **Control Panel Multi-port serial adapters** and select the adapter.

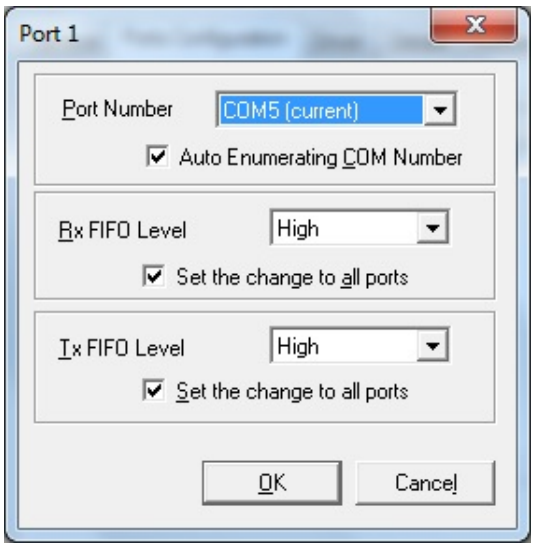
2. Right-click the adapter and select **Properties**.



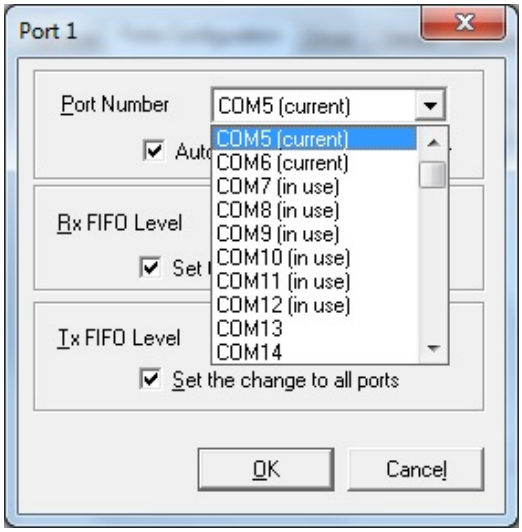
3. Click on the **Port Configuration** tab, select the port, and then click **Port Setting.**



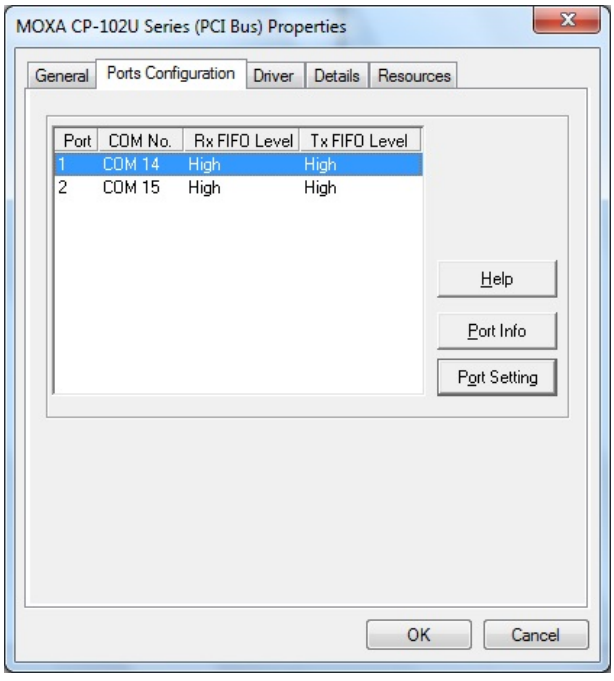
4. To manually change the COM number associated with a port, uncheck **Auto Enumerating COM Number**.



5. Select a new COM port from the list and click **OK**.

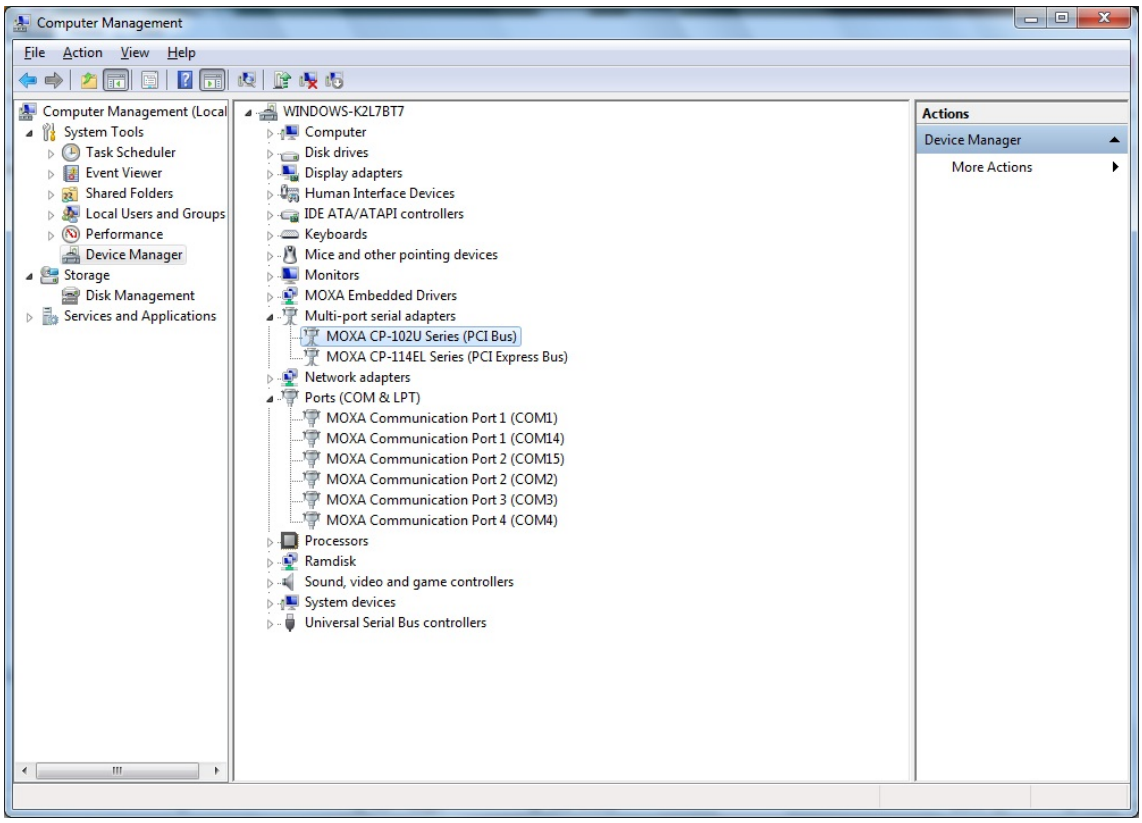


6. Make sure the COM port names are correct, and then click **OK** to activate the settings.



7. To verify that the COM port names have been changed, go to **Ports (COM & LPT)** in the Control Panel.

NOTE Make sure each port name is unique; using duplicate names will result in some devices being inaccessible.



EPM-3032: Changing the Software-Selectable UART Mode

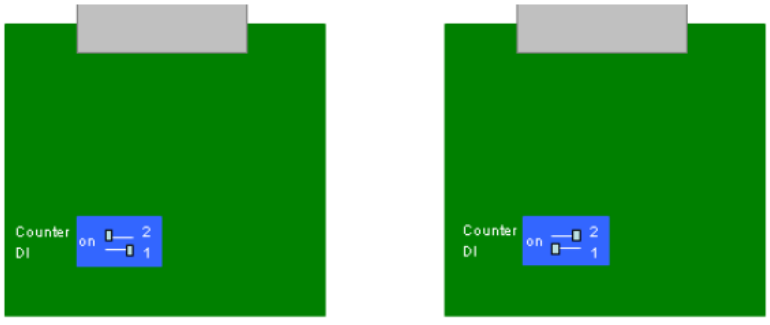
Please refer to the section in Chapter 5: Setting UART Mode section.

EPM-3438: Driver Installation

Before installing the EPM-3438, select counter mode or DI mode for the module.

If dip-switch 1 on the EPM-3438 is on, the DI0 will work in digital input port mode. The DI0 just reflects whether the input signal status is HIGH or LOW. If DIP switch 2 on the EPM-3438 is on, the DI0 works as a 16-bit counter.

The counter is increased when the input pulse is toggled from low to high. See the following figures for DIP switch settings.

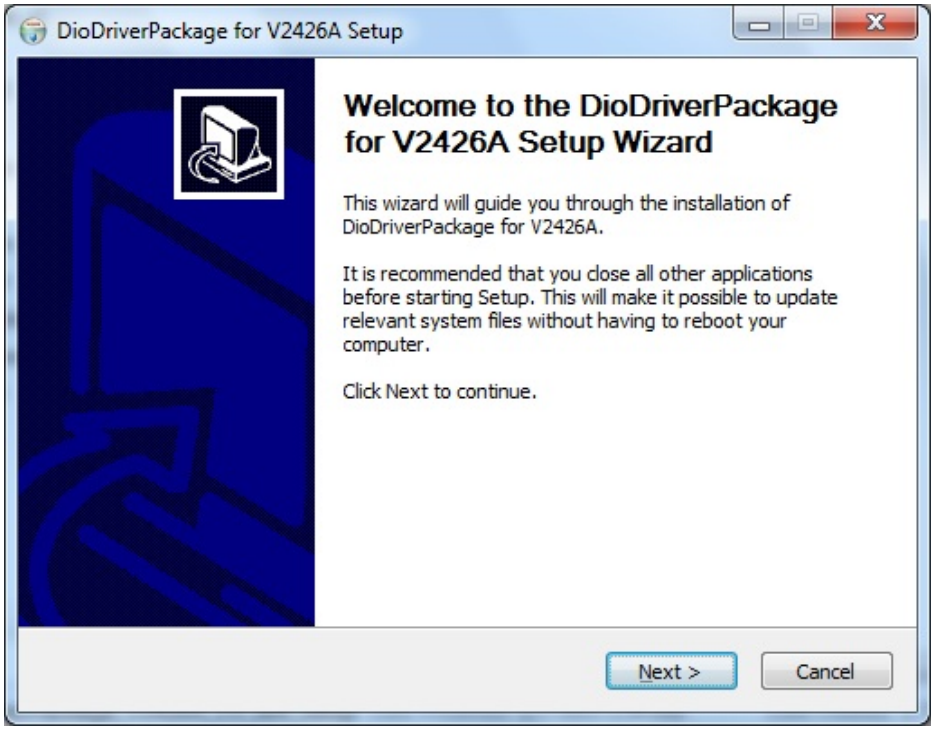


Counter mode DI mode

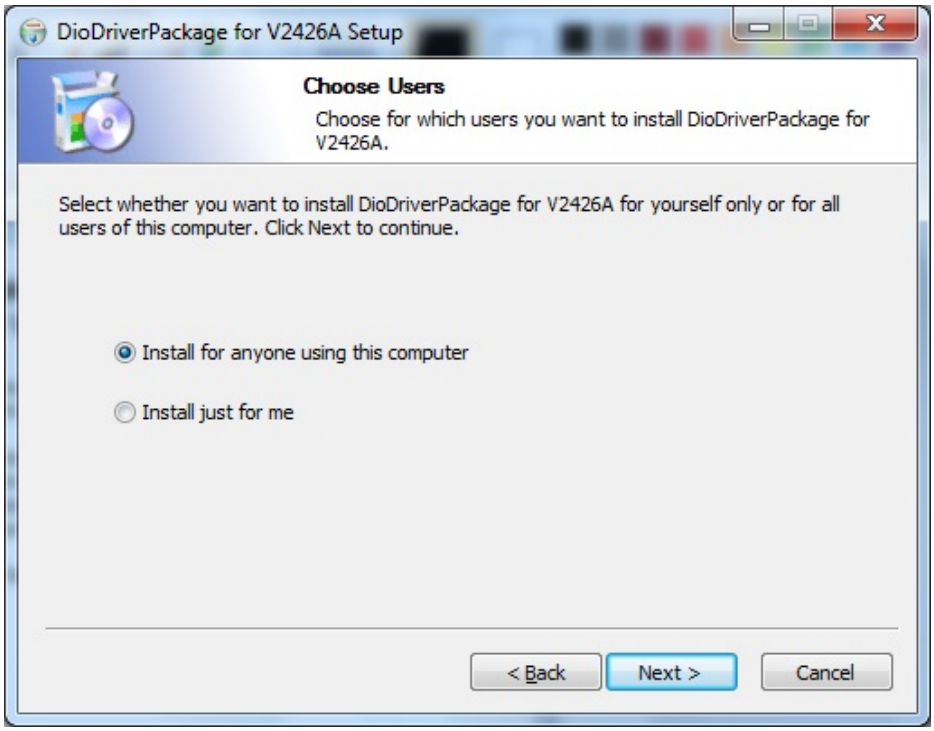
Before using the EPM-3438 expansion module, you need to update the driver. Be sure to install the driver before inserting the expansion module in the slot.

Take the following steps to install the EPM-3438 module driver:

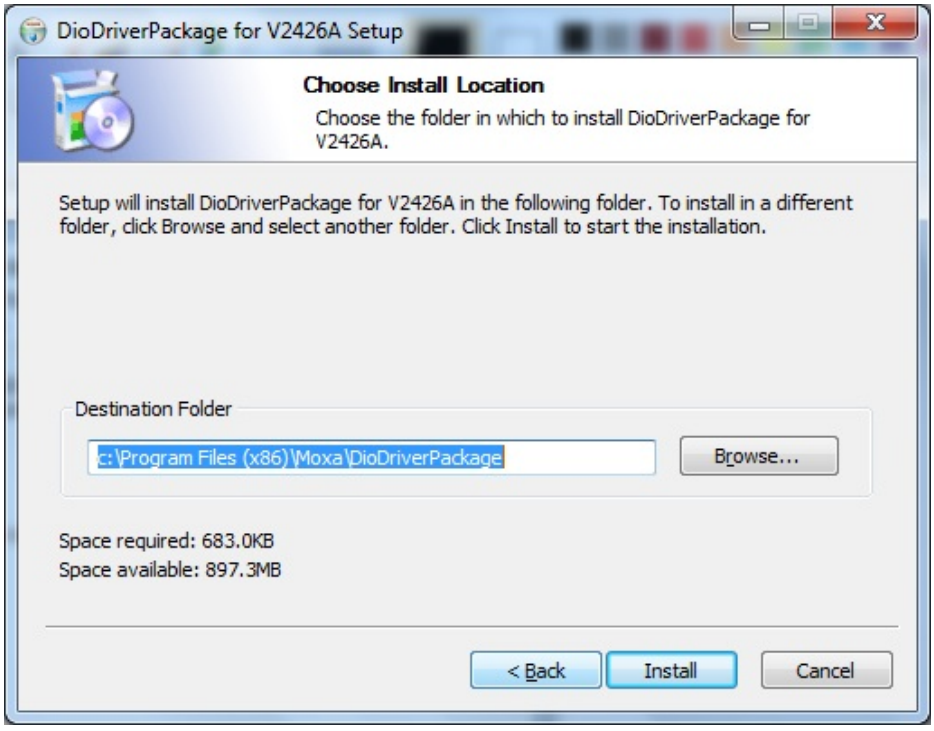
1. Run **DioDriverPackage_V2426A_1.0_x86_Setup.exe** to begin installation and then click **Next**.



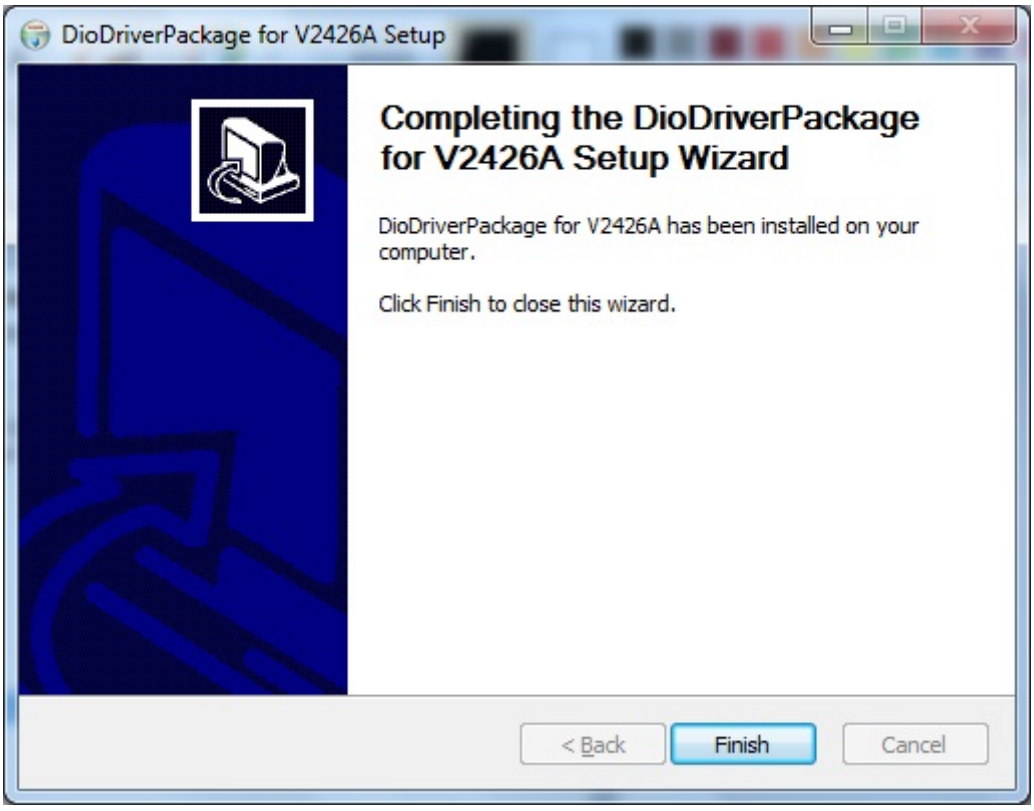
2. Click **Next** to install using default settings.



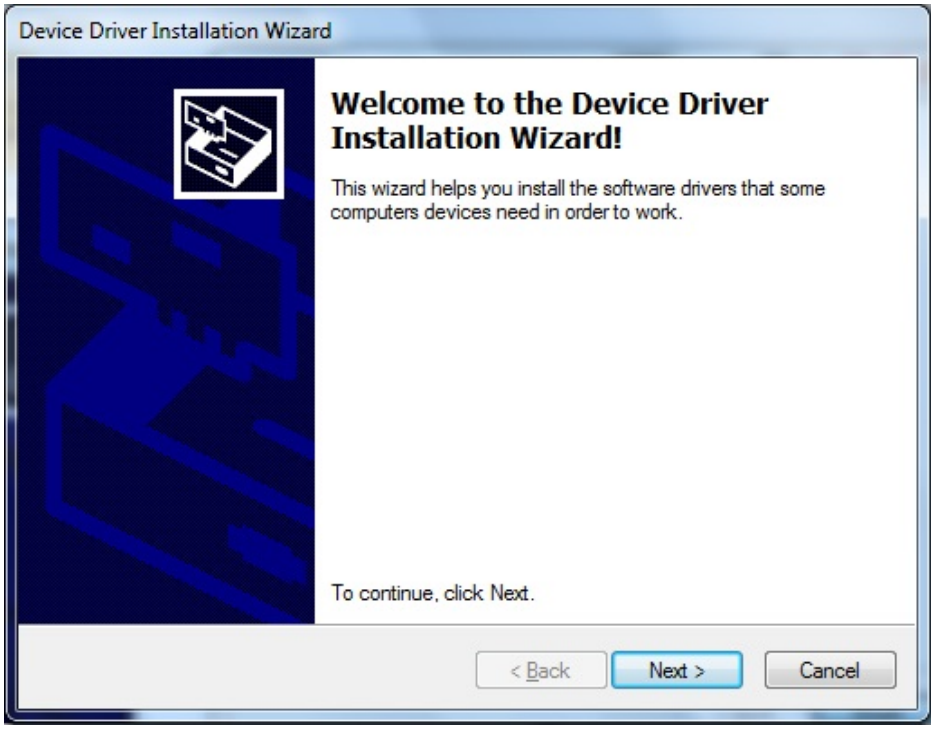
3. Click **Install** to start the installation.



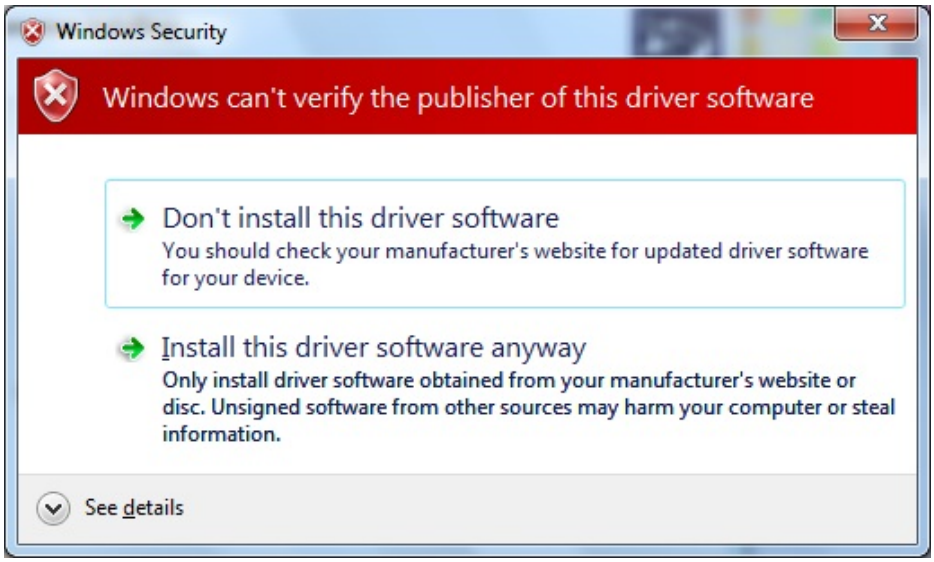
4. Click **Finish**.



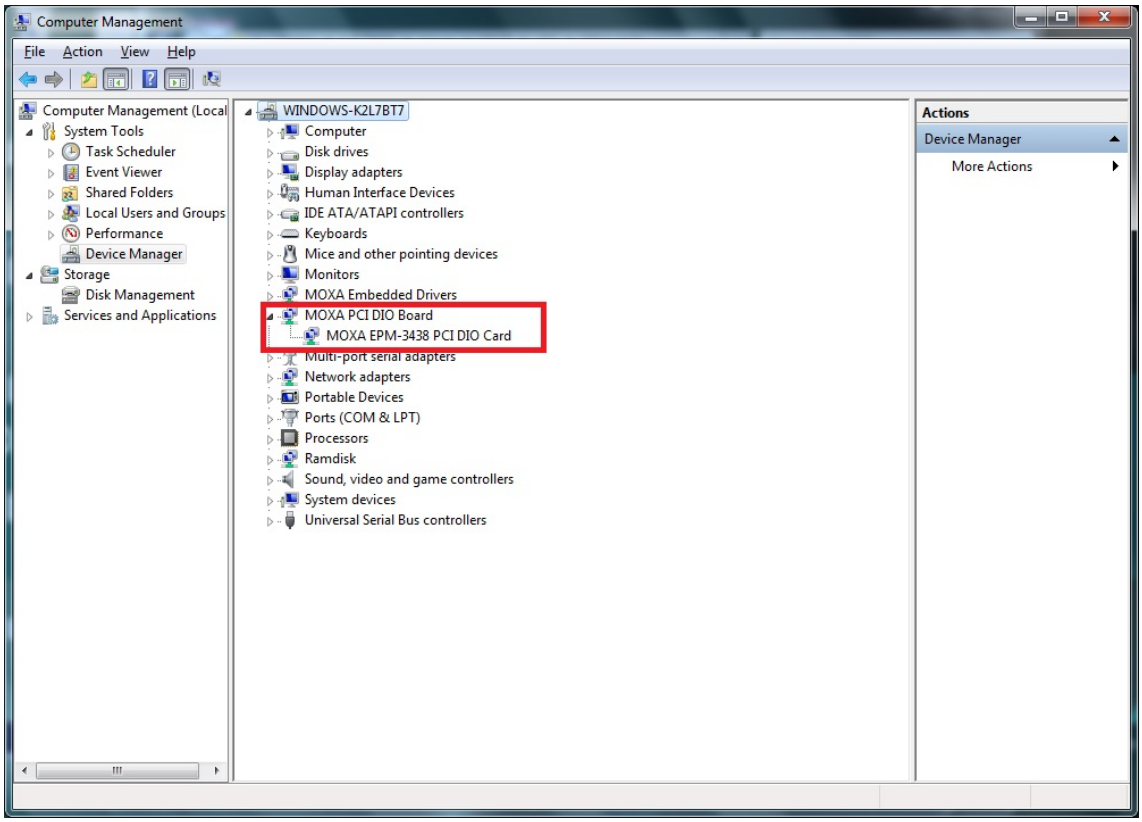
5. Click **Next**.



6. If the warning is exist, just click" Install this driver software anyway"



7. The system should find the new hardware and install the driver automatically. Check the **Windows Device Manager** to verify.



EPM-3438: Programming Guide

Some operations can be configured through programming; the following "DIO" example can be found on the software DVD at **\examples\ExpansionCard\EPM-3438**.

Moxa functions for DI/DO

Function	**HANDLE mxdgio_epm3438_open(int HWIndex);**
Description	This function opens access to the DIO device.
Input	<HWIndex> The first or second EPM-3438 board.
Output	None
Return	When successful, this function returns an access to the DIO device; otherwise, an error.

Function	**void mxdgio_close(HANDLE fd);**
Description	This function closes the access to the DIO device.
Input	<fd> The access to the device.
Output	None
Return	None

Function	**int mxdgio_get_input_signal(HANDLE fd, int port);**
Description	This function gets the signal state of a digital input channel.
Input	<fd> The access to the device. <port> port #
Output	<state> DIO_HIGH (1) for high, DIO_LOW (0) for low
Return	Returns 1 for a high signal or 0 for a low signal, if successful. Otherwise, it returns a value of -1.

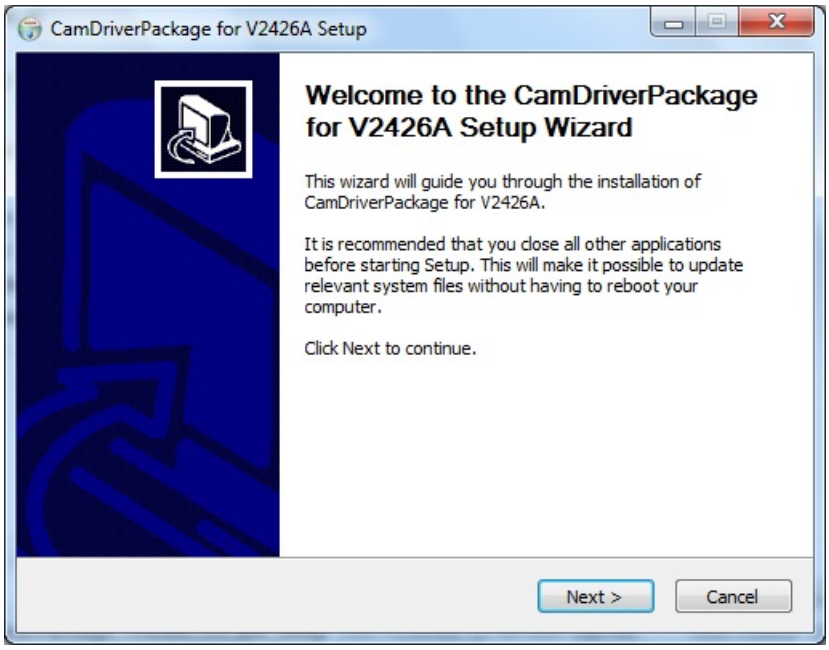
Function	**int mxdgio_get_output_signal(HANDLE fd, int port);**
Description	This function gets the signal state of a digital output channel.
Input	<fd> The access to the device. <port> Port number
Output	None
Return	Returns 1 for a high signal or 0 for a low signal, if successful. Otherwise, it returns a value of -1.

Function	**int mxdgio_set_output_signal_high(HANDLE fd, int port);**
Description	This function sets a high signal to a digital output channel.
Input	<fd> The access to the device. <port> Port number.
Output	none.
Return	When successful, this function returns 0. When an error occurs, it returns -1.

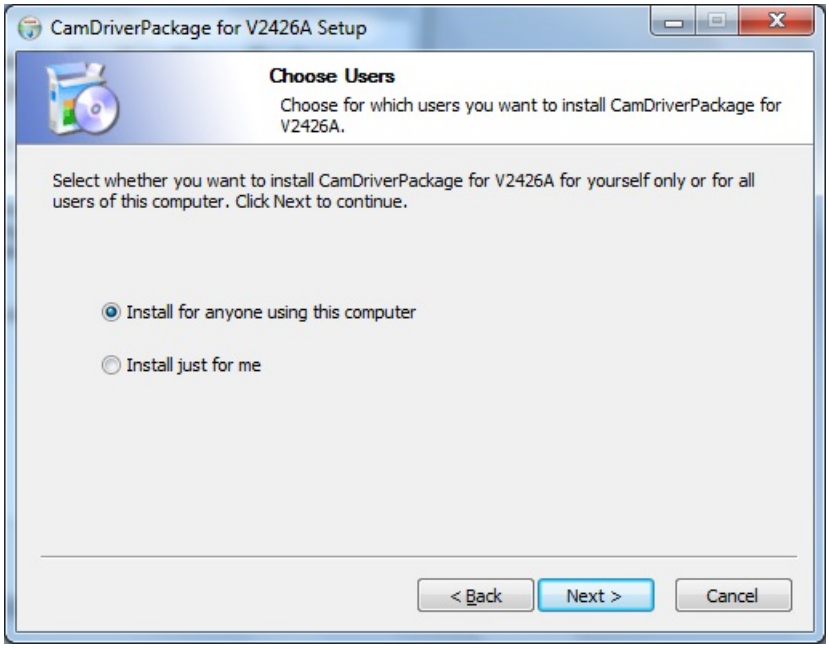
Function	**int mxdgio_set_output_signal_low(HANDLE fd, int port);**
Description	This function sets a low signal to a digital output.
Input	<fd> The access to the device. <port> Port number.
Output	none.
Return	When successful, this function returns 0. When an error occurs, it returns -1.

EPM-3112: Driver Installation

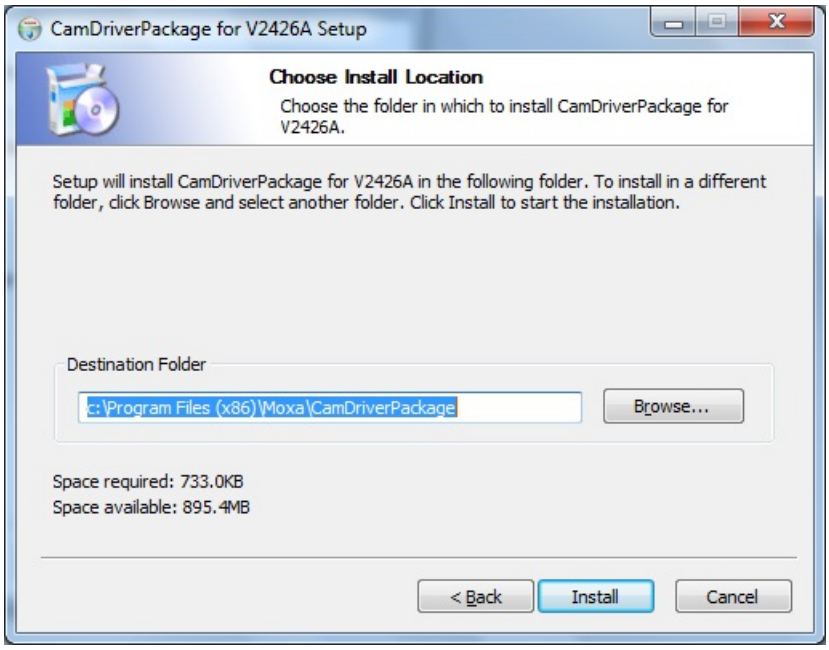
1. Run **CamDriverPackage_V2426A_1.0_x86_Setup.exe** to begin installation and then click **Next**.



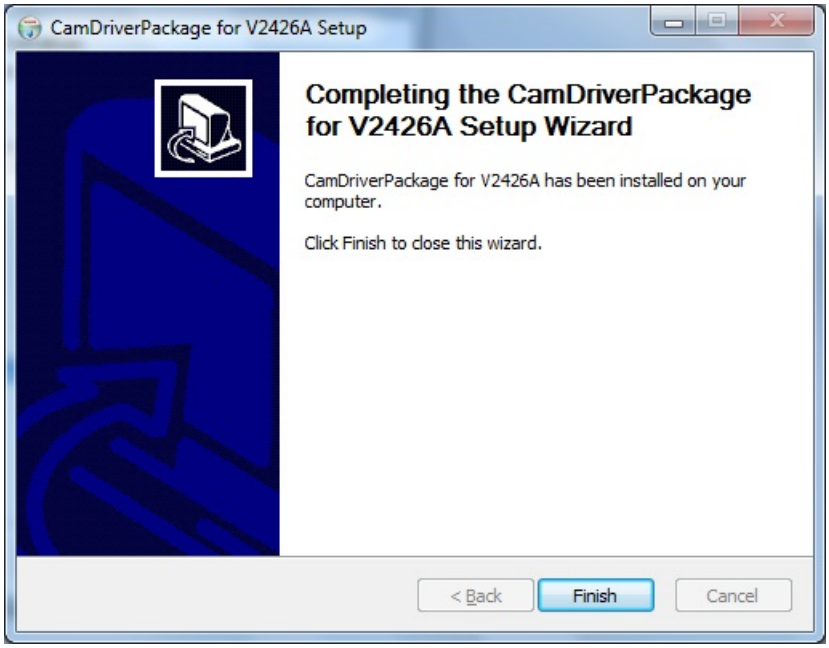
2. Click **Next** to install using default settings.



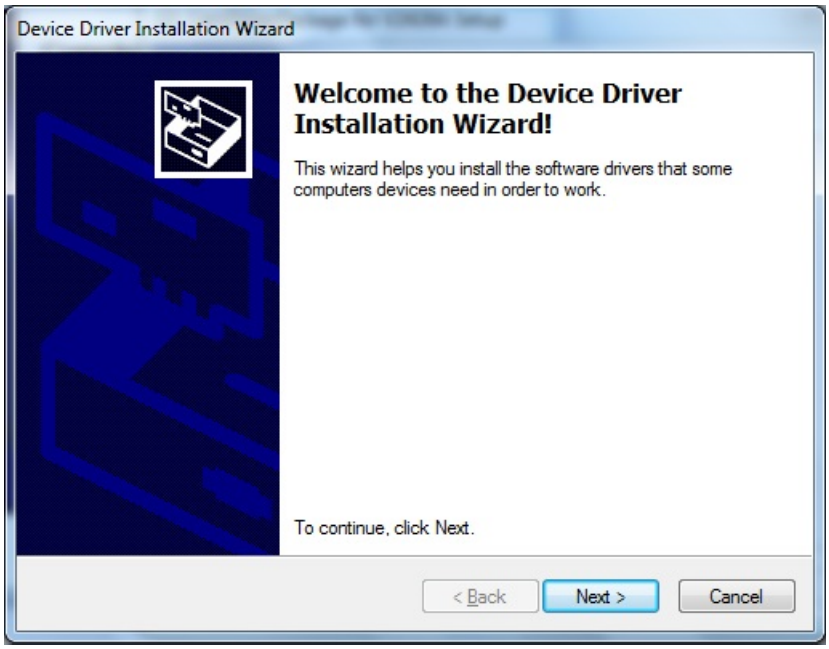
3. Click **Install** to start the installation.



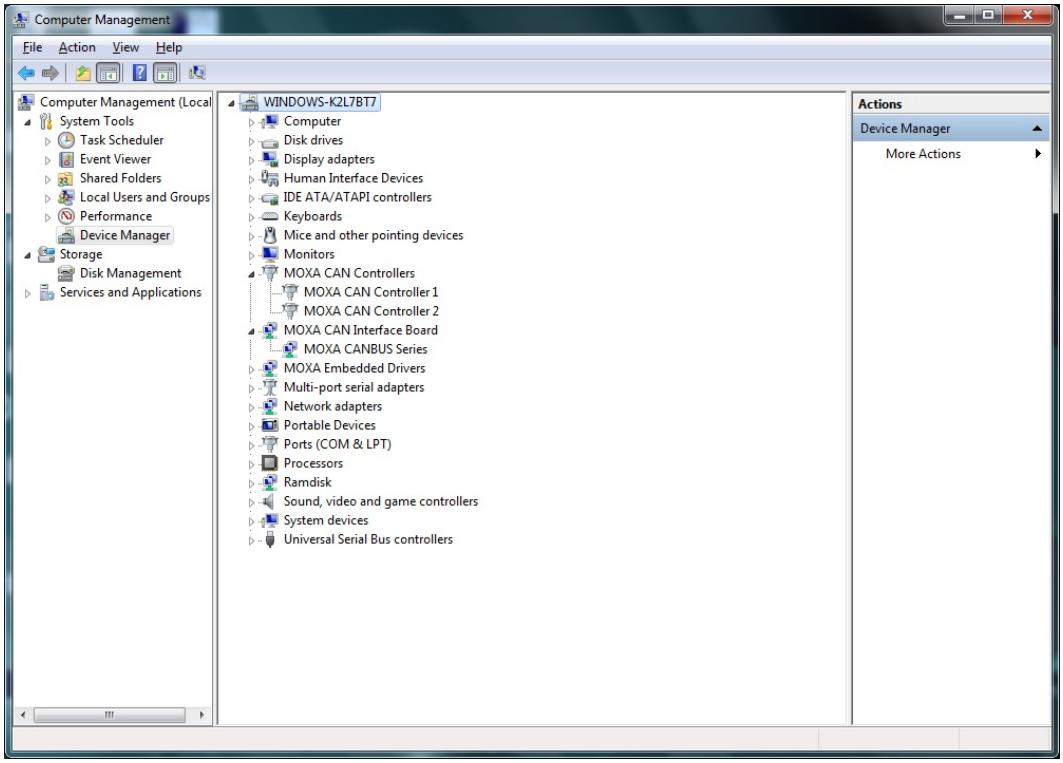
4. Click **Finish**.



5. Click **Next**.



6. The system should find the new hardware and install the driver automatically. Check the **Windows Device Manager** to verify.



EPM-3112: Programming Guide

Some operations can be configured through programming; the following "CANBUS" example can be found on the software DVD at **\examples\EPM-3112**.

Moxa CAN Bus Library

int mxcan_close (int fd)	
Description	Close an open port.
Input	<fd> the open port
Return Value	None

unsigned int mxcan_get_bus_timing (int fd)	
Description	Gets the bus timing of an open port.
Input	<fd> the open port
Return Value	0 on failure, otherwise the bus speed in KHz

int mxcan_get_parameters (int fd, CANPRM * param)	
Description	Gets the parameter of an open port.
Input	<fd> the open port
Output	< param> pointer to the CANPRM structure
Return Value	0 on failure, otherwise returns a negative value

int mxcan_get_registers (int fd, unsigned char * buffer, int num)	
Description	Gets the register values of an open port.
Input	<fd> the open port
Output	< buffer > pointer to a buffer for these values <num> number of register values; for a module with sja1000 chipset, the value must be 32
Return Value	0 on success; other numbers indicate failure

int mxcan_get_stat (int fd, CANBST * stat)	
Description	Gets the statistics of an open port.
Input	<fd> the open port
Output	< stat > pointer to a container of the statistics
Return Value	0 on success; other numbers indicate failure

int mxcan_inqueue (int fd)	
Description	Gets the number of received bytes that are queued in the driver of an open port.
Input	<fd> the open port
Return Value	0 on failure; otherwise the number of bytes

int mxcan_open (int port)	
Description	Open a can port given the port number.
Input	<port> port number starting from 1; in Linux, open port 1 will open /dev/can0
Return Value	-1 on failure; otherwise returns fd

int mxcan_outqueue (int fd)	
Description	Gets the number of bytes waiting to be transmitted to a CAN port.
Input	<fd> the open port
Return Value	-1 on failure; otherwise the number of bytes

int mxcan_purge_buffer (int fd, unsigned int purge)	
Description	Purges the buffers of an open port.
Input	<fd> the open port
Output	< purge> 1: received data buffer; 2: transmit data buffer; otherwise: both
Return Value	0 on success; otherwise failure

int mxcan_read (int fd, char * buffer, int size)	
Description	Reads data into a buffer from an open port (the size should be a multiple of the CANMSG size)
Input	<fd> the open port
Output	<buffer> pointer to the buffer
Return Value	0 on failure (data not available); otherwise the number of bytes read

int mxcan_set_bus_timing (int fd, unsigned int speed)	
Description	Sets the bus timing of an open port.
Input	<fd> the open port
Output	<speed> bus timing in Hz
Return Value	0 on success; otherwise returns a negative number

int mxcan_set_nonblocking (int fd)	
Description	Sets the open fd to be non-blocking.
Input	<fd> the open port
Return Value	0 on success; otherwise returns a negative number

int mxcan_set_parameters (int fd, CANPRM * param)	
Description	Sets the parameters of an open port.
Input	<fd> the open port <param> pointer to the CANPRM structure
Output	<speed> bus timing in Hz
Return Value	0 on success; otherwise returns a negative number

int mxcan_set_read_timeout (int fd, unsigned int to)	
Description	Sets data reading timeout of an open port.
Input	<fd> the open port <to> timeout in milliseconds
Return Value	0 on success; otherwise failure

int mxcan_set_write_timeout (int fd, unsigned int to)	
Description	Sets data writing timeout of an open port.
Input	<fd> the open port <to> timeout in milliseconds
Return Value	0 on success; otherwise failure

int mxcan_write (int fd, char * buffer, int size)	
Description	Writes data to the open port
Input	<fd> the open port <buffer> pointer to the data <size> size of the data (should be a multiple of the CANMSG size)
Return Value	0 on failure; otherwise the number of bytes written

EPM-DK02: Driver Installation

Please refer to these sections in Chapter 4:

Wi-Fi Module Driver Installation
LTE Module Driver Installation
3G Module Driver Installation
3G-GPS Module Driver Installation

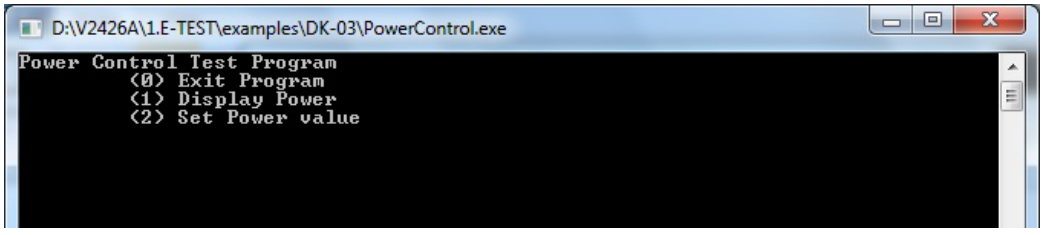
EPM-DK02: Controlling Power

The EPM-DK02 module provides a power control function that lets you control the power of a USB device so that you can enable or disable the device. This section introduces how to configure this function to enable/disable a USB DOM.

Note that the power on/off control function is only suitable for devices that have a USB interface. If you are using a device with a PCIe interface, do not enable the power on/off control function, since doing so could damage the device.

EPM-DK02: Getting the USB Sockets' Current Power Status

1. Execute **PowerContorl.exe**, located on the CD-ROM at **\examples\DK-03\PowerContorl\Release**.

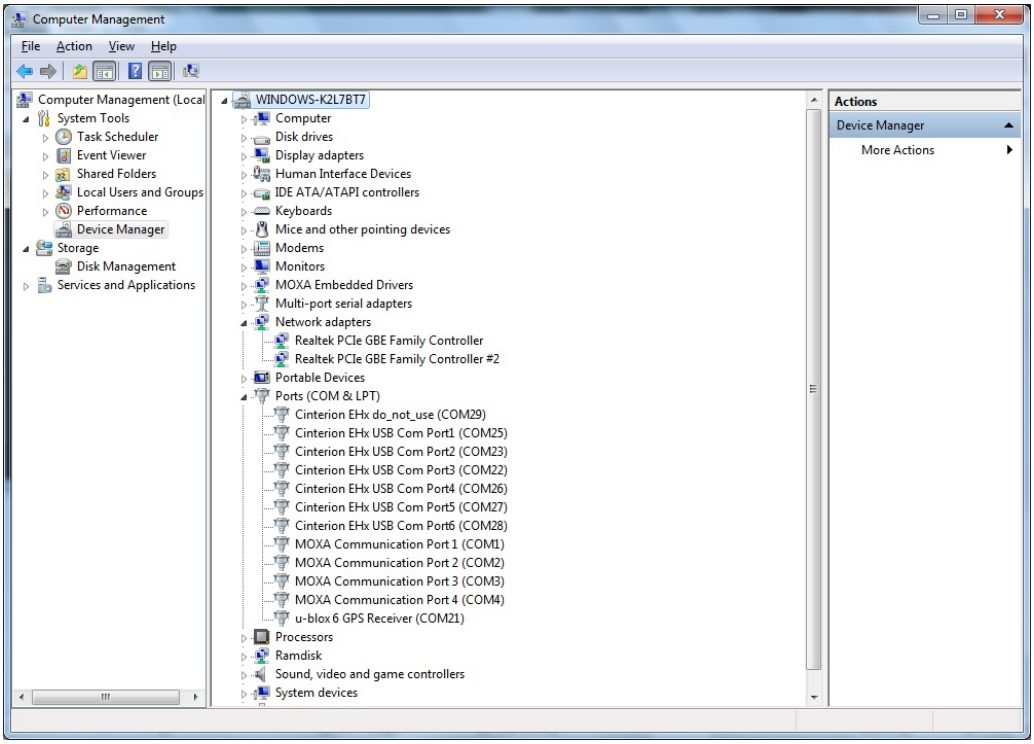


2. You may verify that the current power status for your selected devices matches that reported by the utility by checking the **Windows Device Manager**.

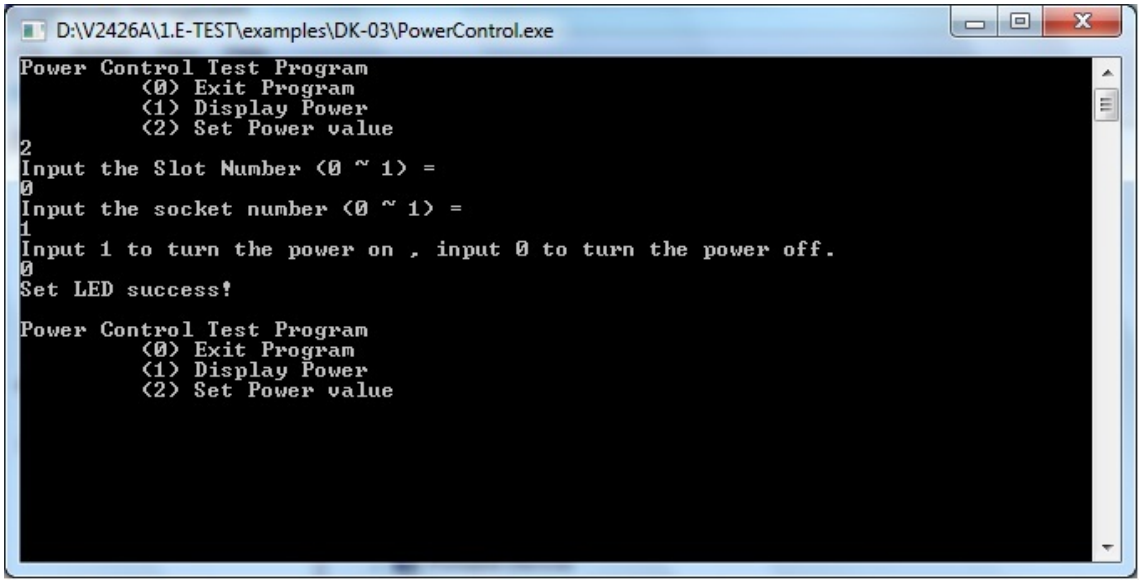
EPM-DK02: Enabling/Disabling USB Socket Power

Take the following steps to disable the USB power-on socket:

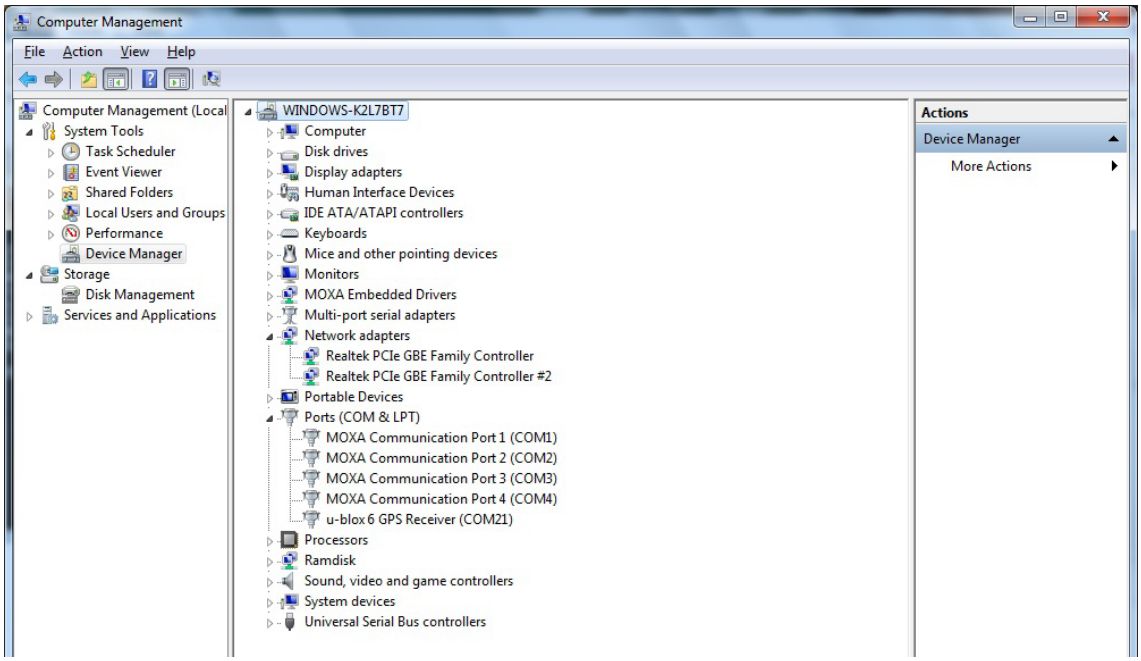
1. Execute **PowerContorl.exe** and check the **Windows Device Manager**.



2. Select the socket and change the status.



3. Check the **Windows Device Manager** to see if the device is disabled.

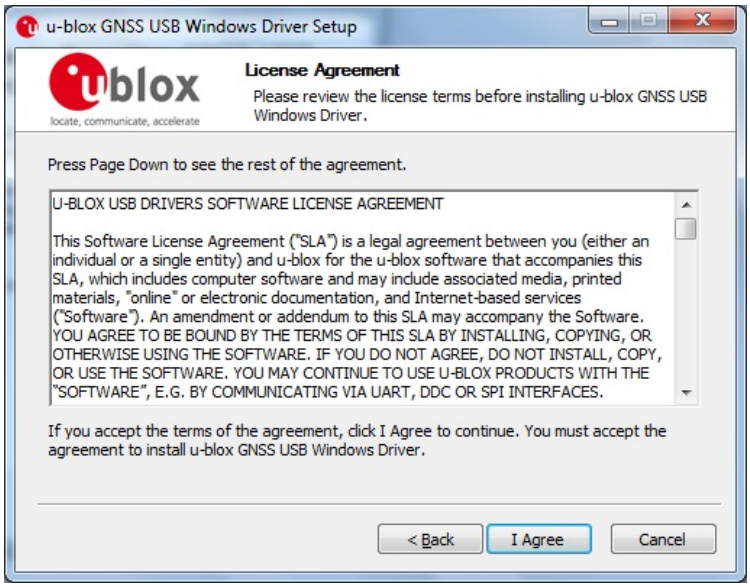


EPM-DK03: GPS Driver Installation

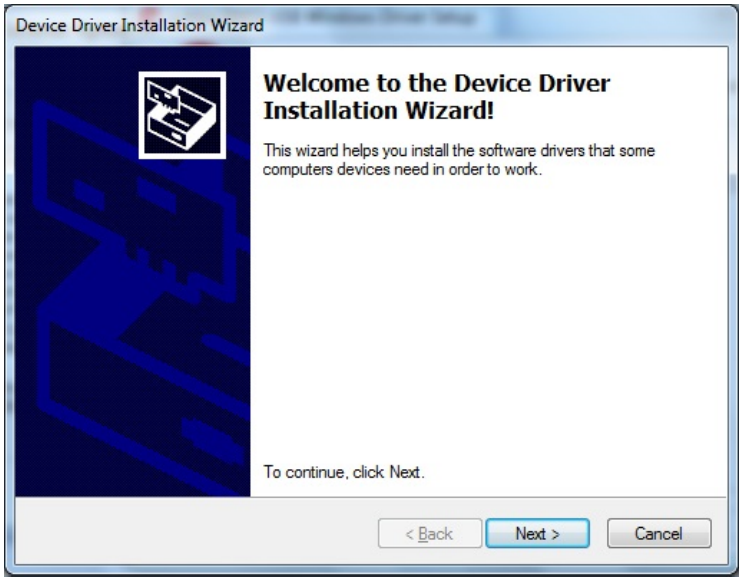
1. Run **ubloxGnss_usbcdc_windows_3264_v1.2.0.8.exe**, select language and click **OK**

2. Click **Next**.

3. Read the license and click "I agree".

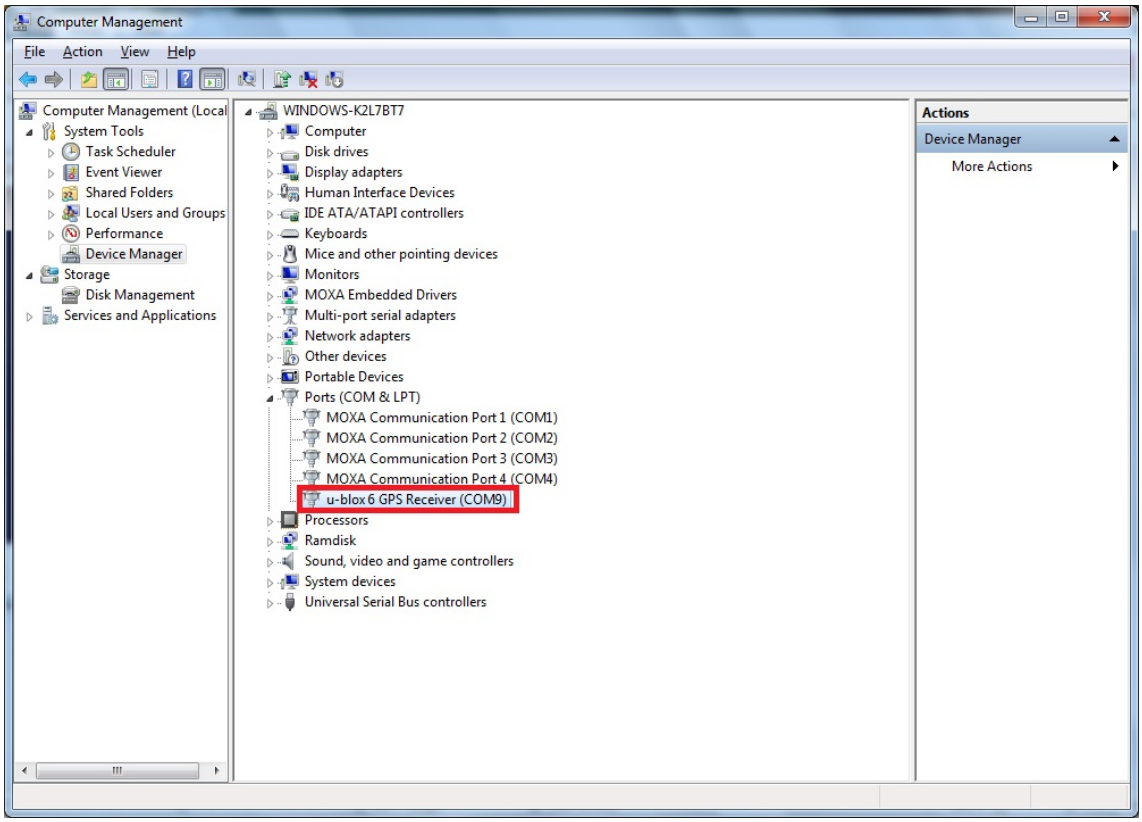


4. Click **Next**.

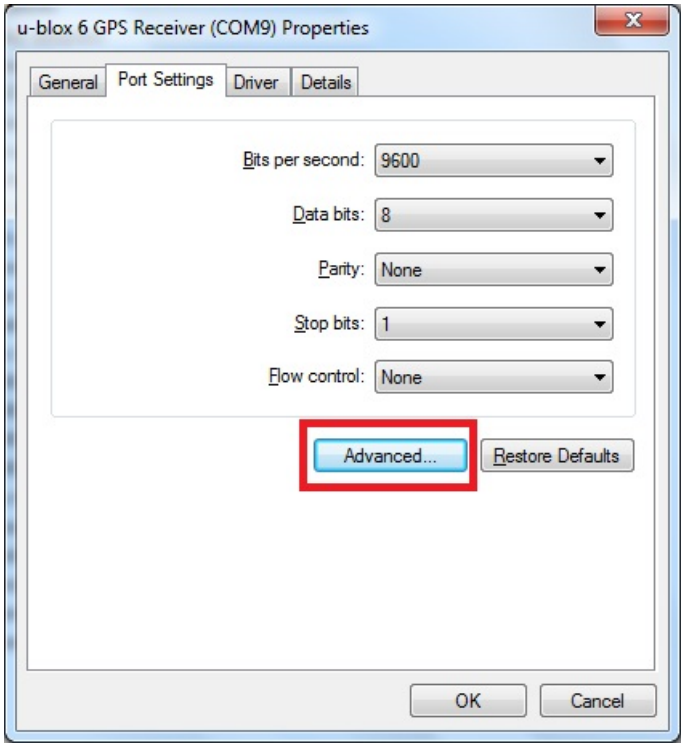


5. Click **Finish**.

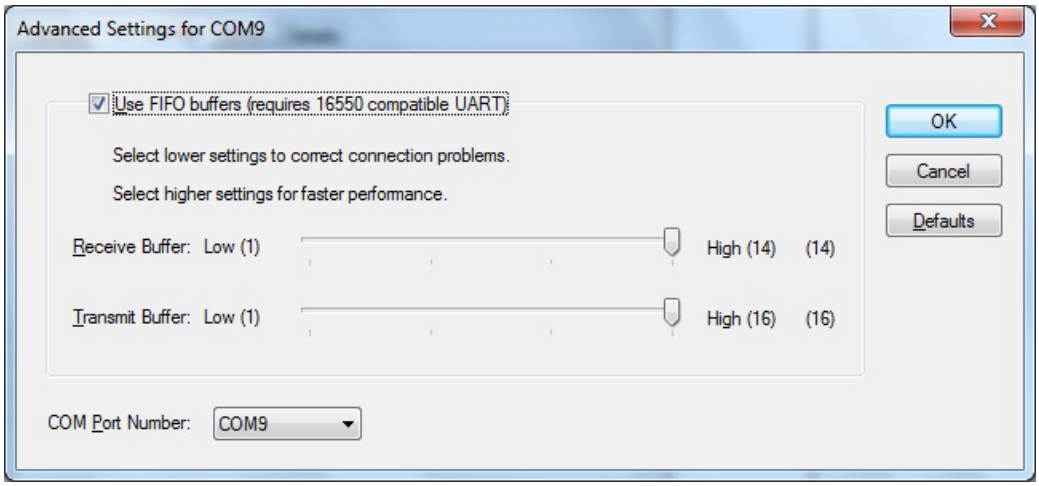
6. The system should find the new hardware and install the driver automatically. Check the **Windows Device Manager** to verify.



7. To change the GPS receiver's port number select **Advanced...** under the **Port Settings** tab.



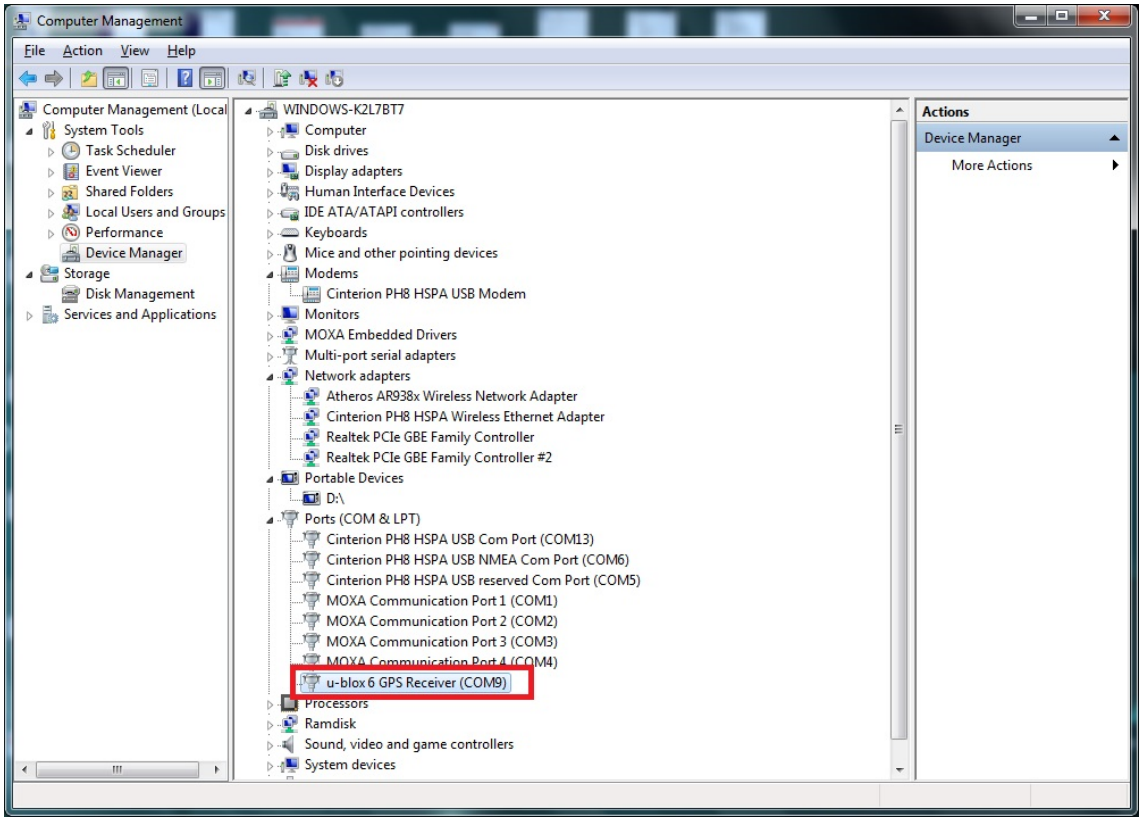
8. Select your desired COM port number from the drop-down menu in the lower. Click **OK** to finish.



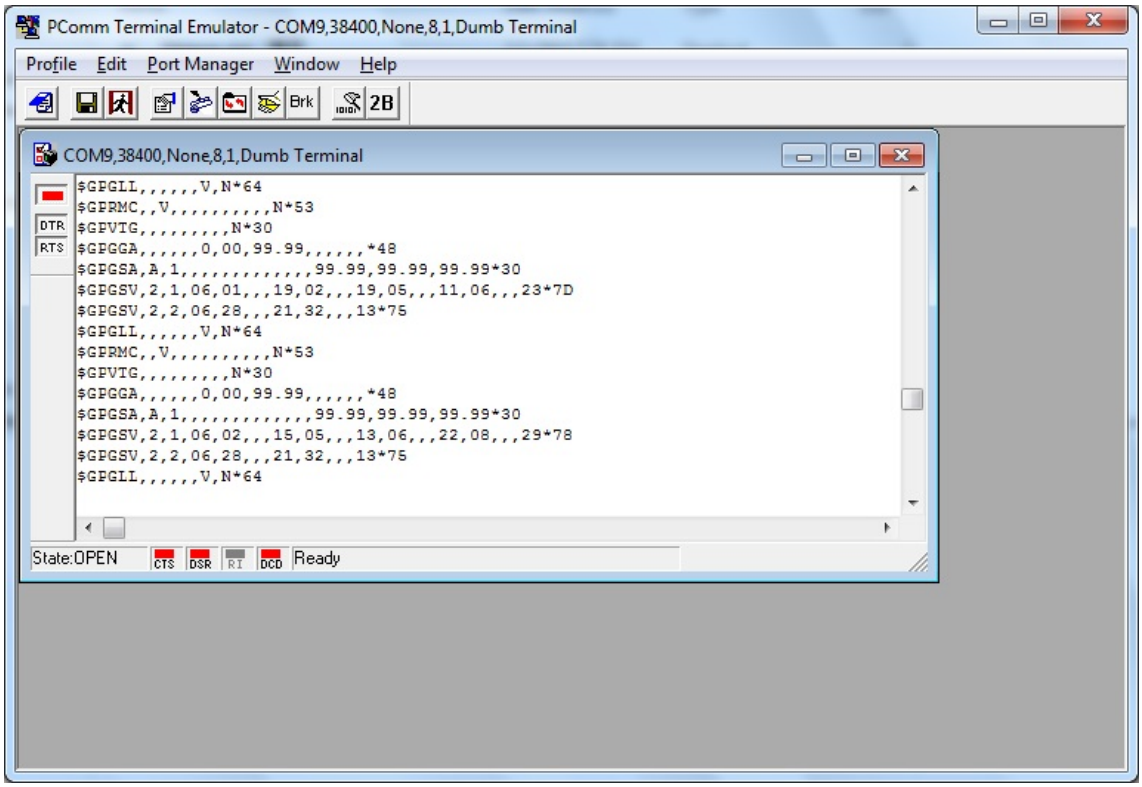
9. You may now use a serial terminal like **Windows HyperTerminal** to view GPS data

EPM-DK03: GPS Module Configuration

1. Start the **Device Manager** and check the "u-blox 6 GPS Receiver" COM port, then start **Mxterm** utility

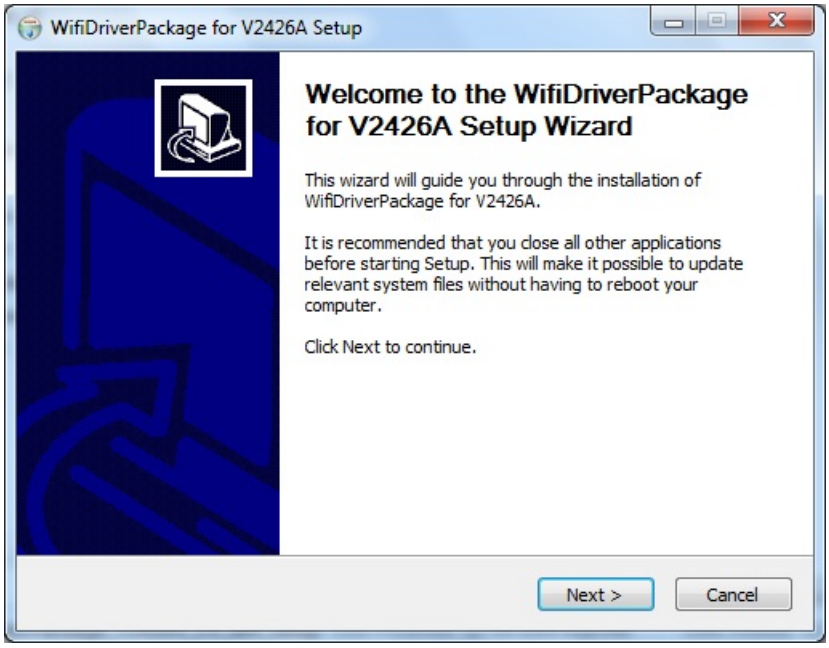


2. You may view the **information** returned by the GPS in "u-blox 6 GPS Receiver" port and verify that the device is working properly correct.

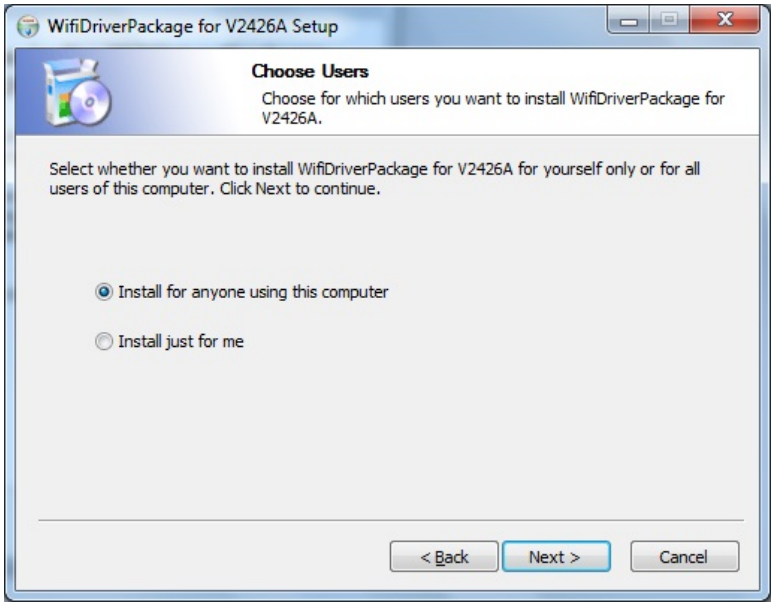


Wi-Fi Module Driver Installation

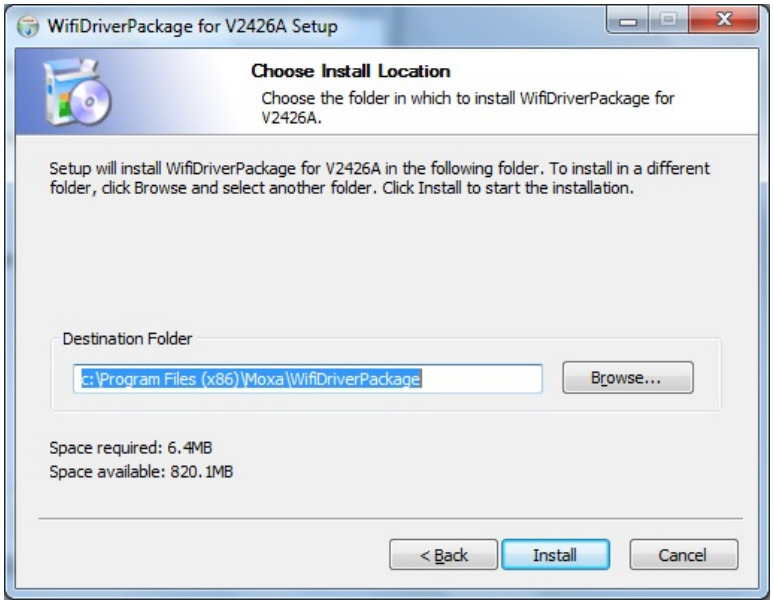
1. Run **WifiDriverPackage_V2426A_1.0_x86_Setup.exe** to begin installation and then click **Next**.



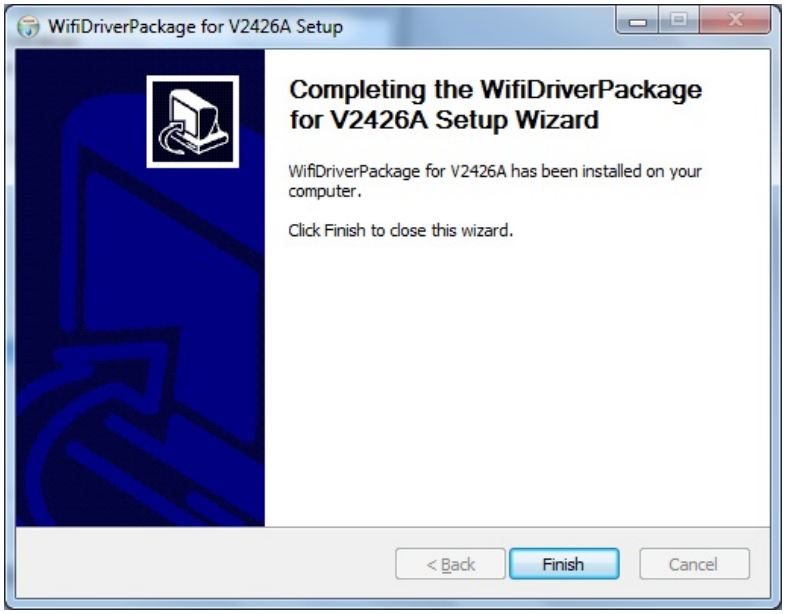
2. Click **Next** to install using default settings.



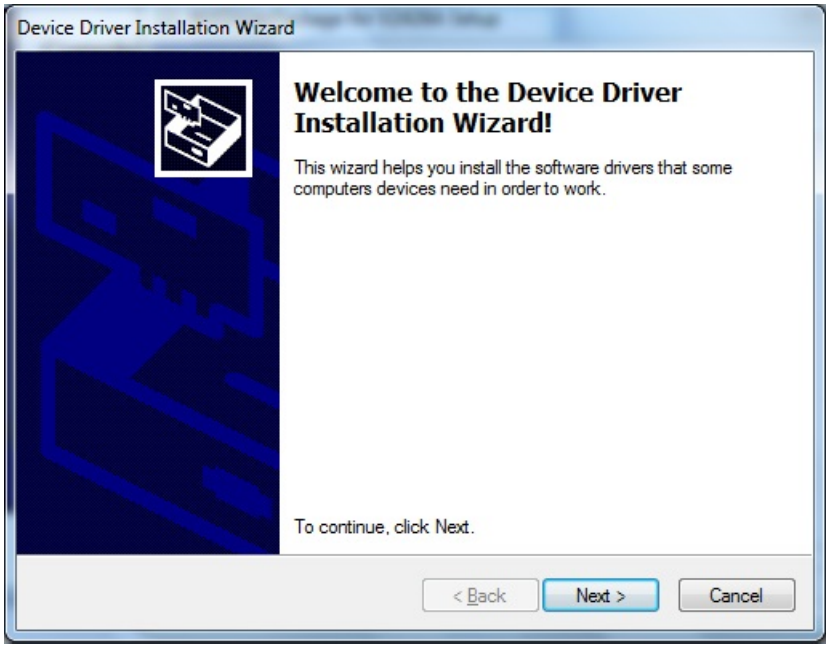
3. Click **Install** to start the installation.



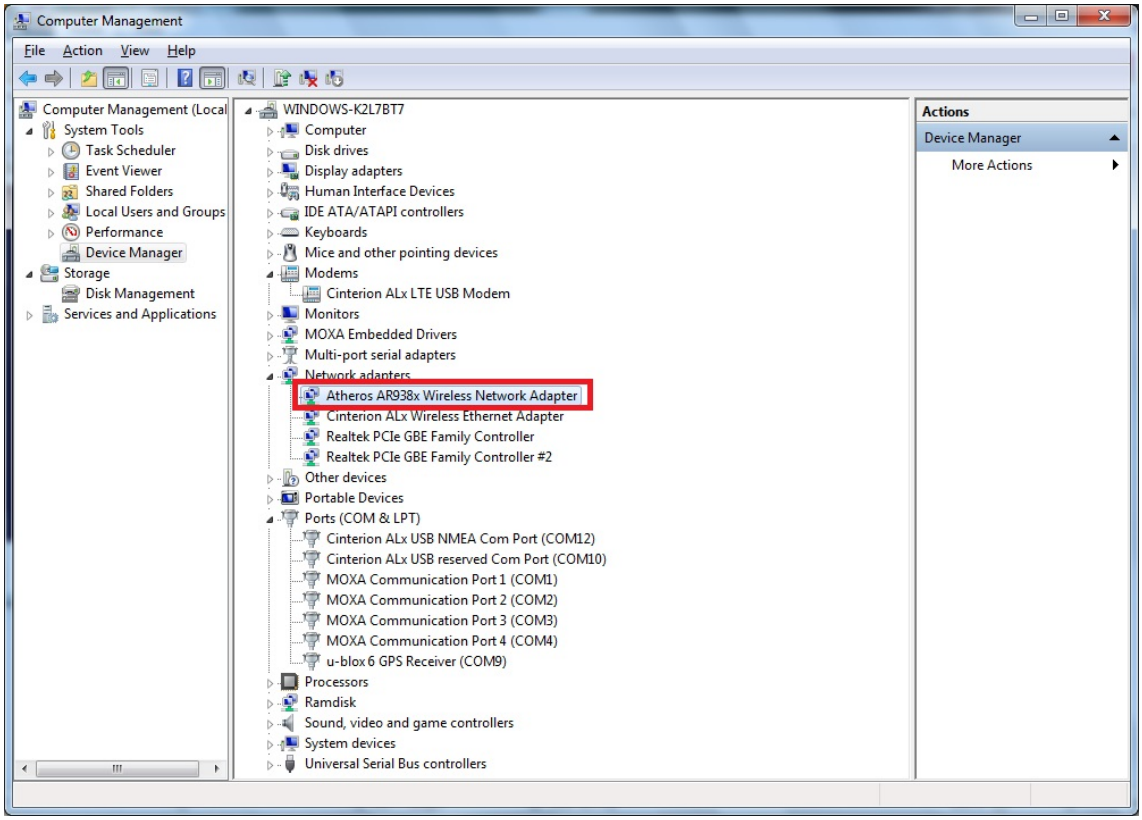
4. Click **Finish**.



5. Click **Next**.

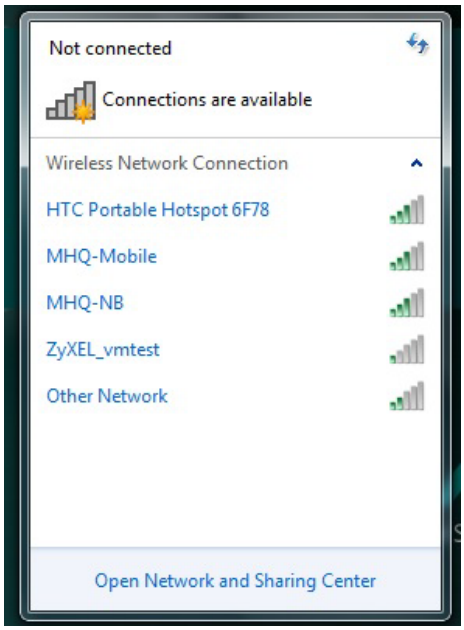


6. The system should find the new hardware and install the driver automatically. Check the **Windows Device Manager** to verify.



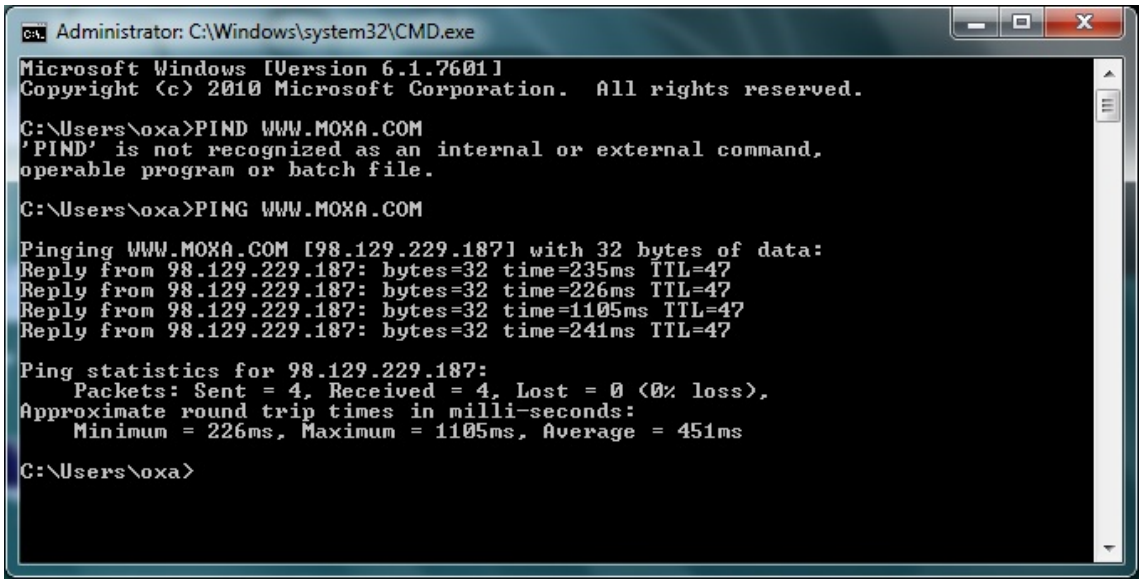
Wi-Fi Module Driver Configuration

1. After installing Wi-Fi module driver, Select wireless network connection from taskbar.



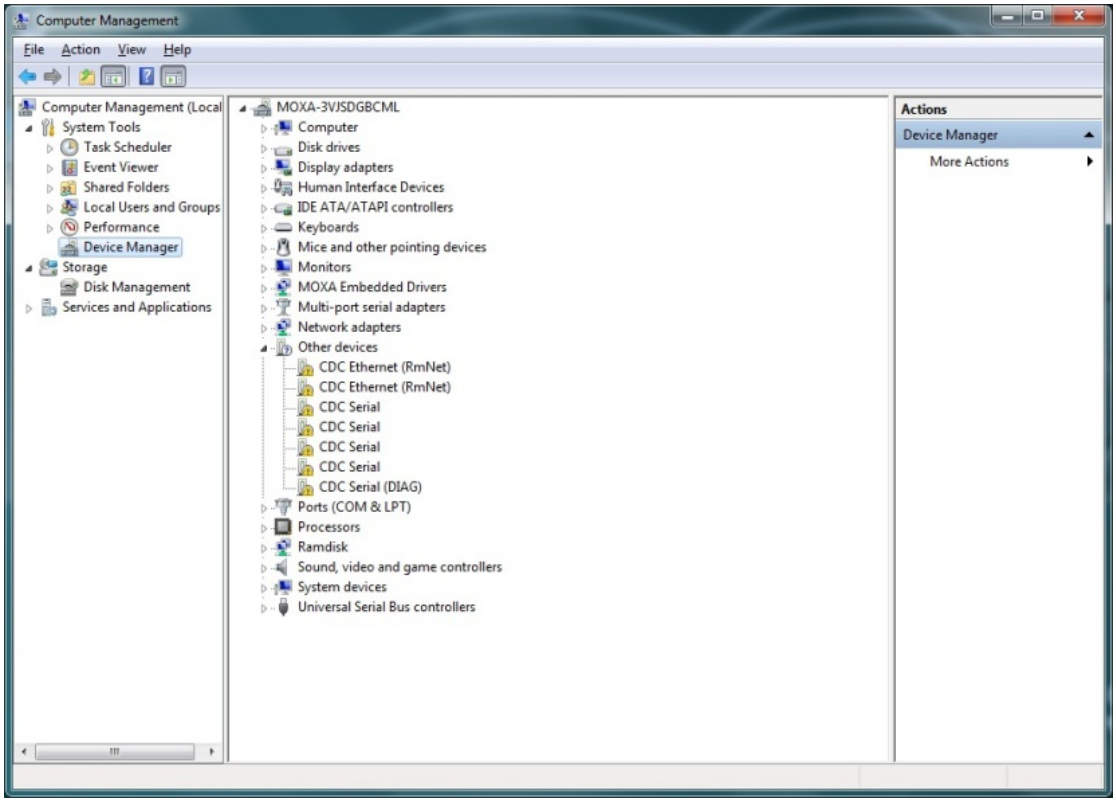
2. Enter the password then you can connect network.

3. You may verify the cellular connection by pinging the interface once the connection is established.

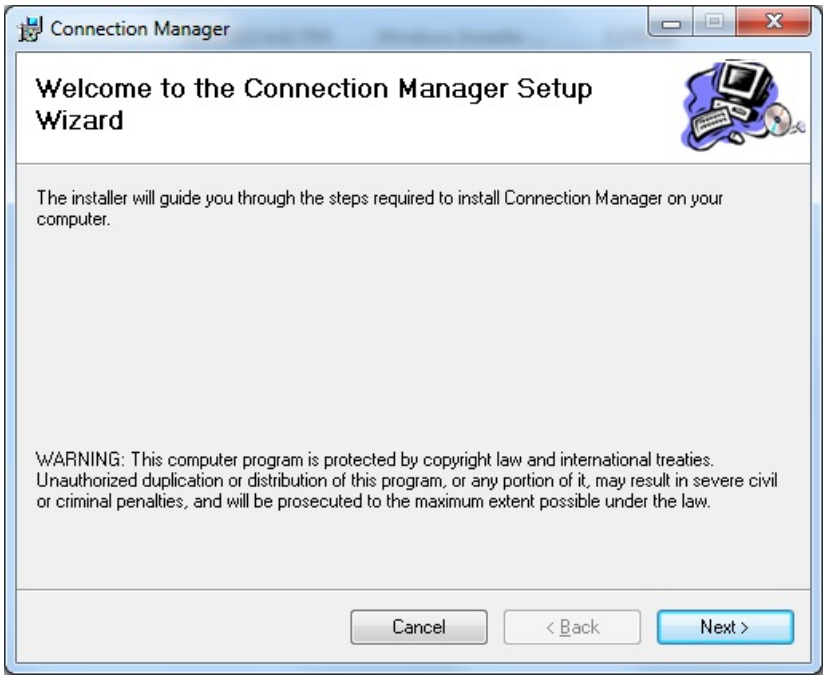


LTE Module Driver Installation

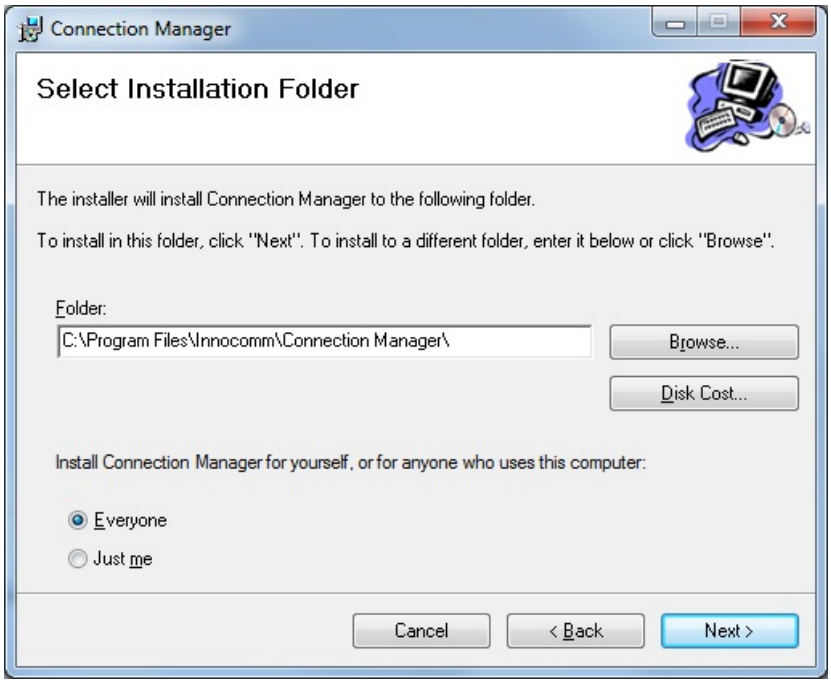
1. Open Windows **Device Manager** to check that the LTE module has been detected.



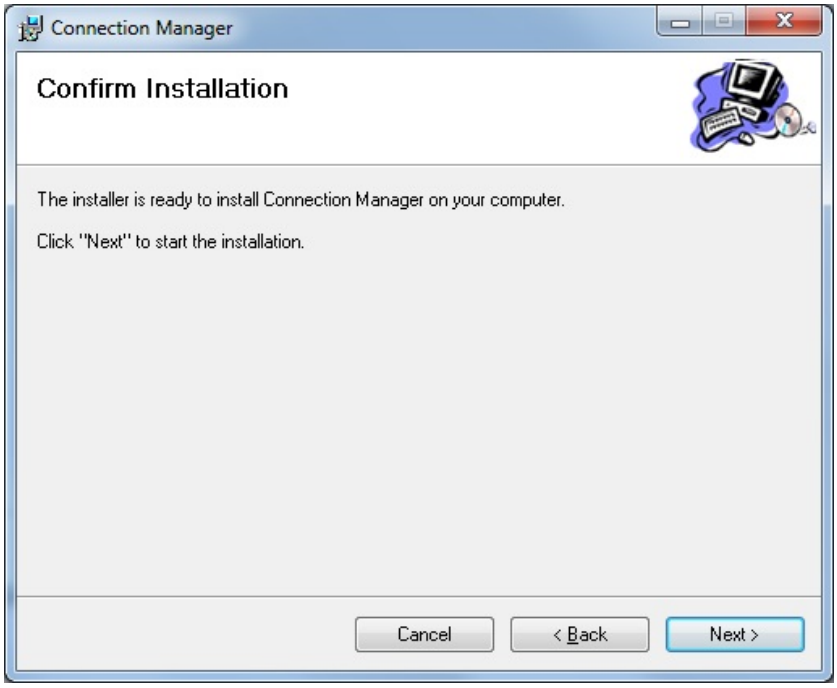
2. Run **iComSetup.exe** from the software CD (path: CD_PATH\utility\innocomm AP) to install the Connection Manager and click **Next**.



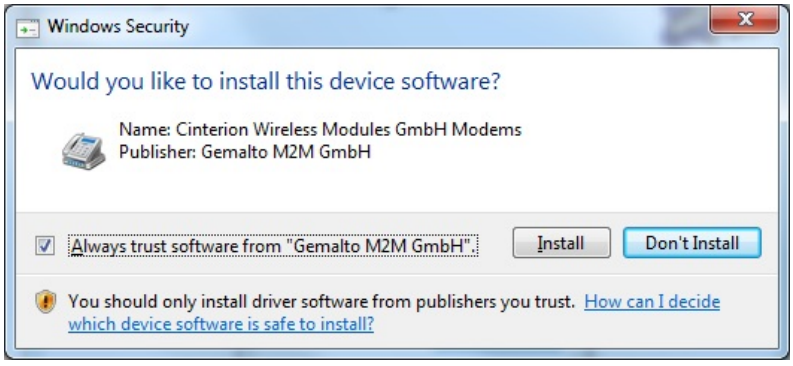
3. Browse to the folder that you want the driver to be installed in and click **Next**.



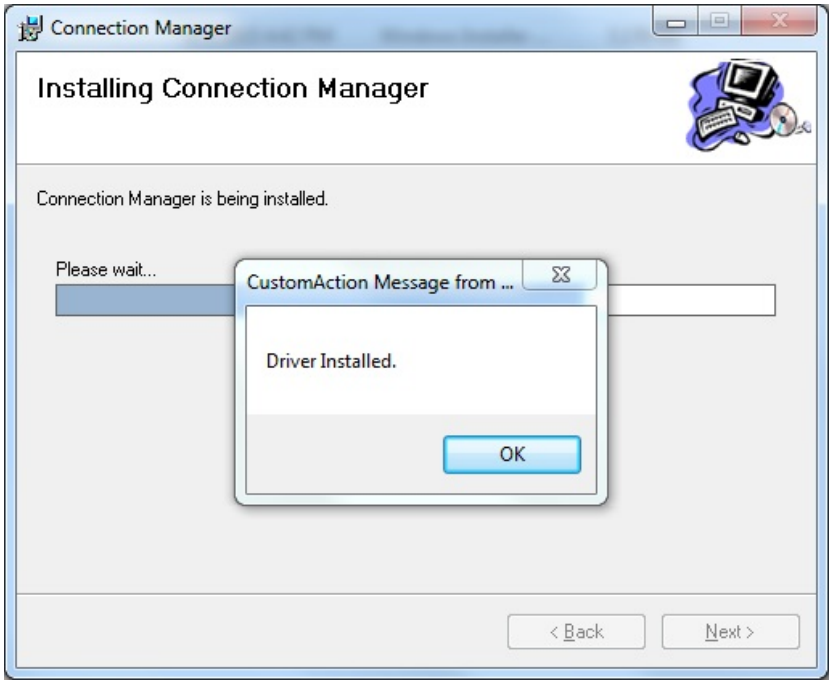
4. Click **Next** again to confirm the installation process.



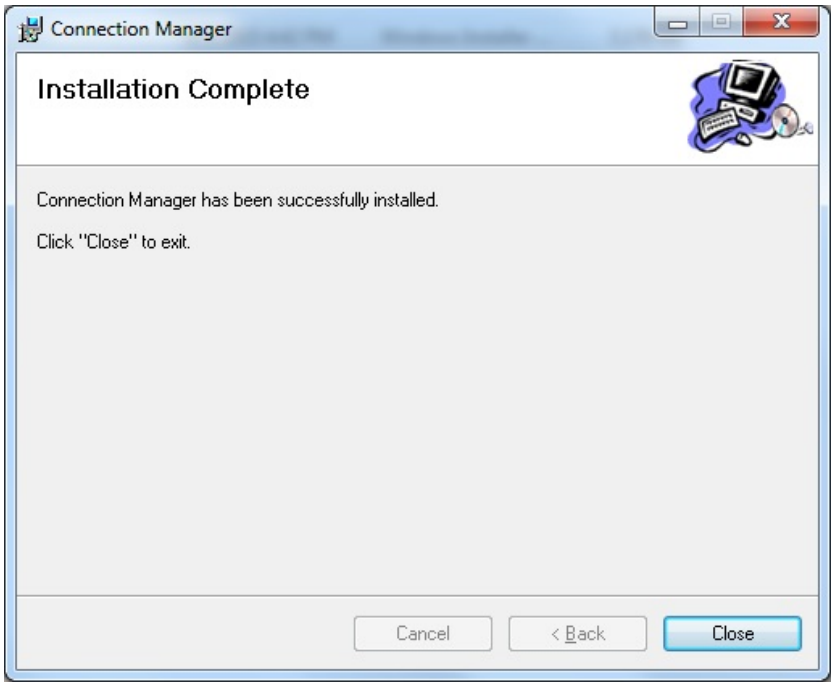
5. Select the **Always trust software from "Gemalto M2M GmbH"** option and click **Install.**



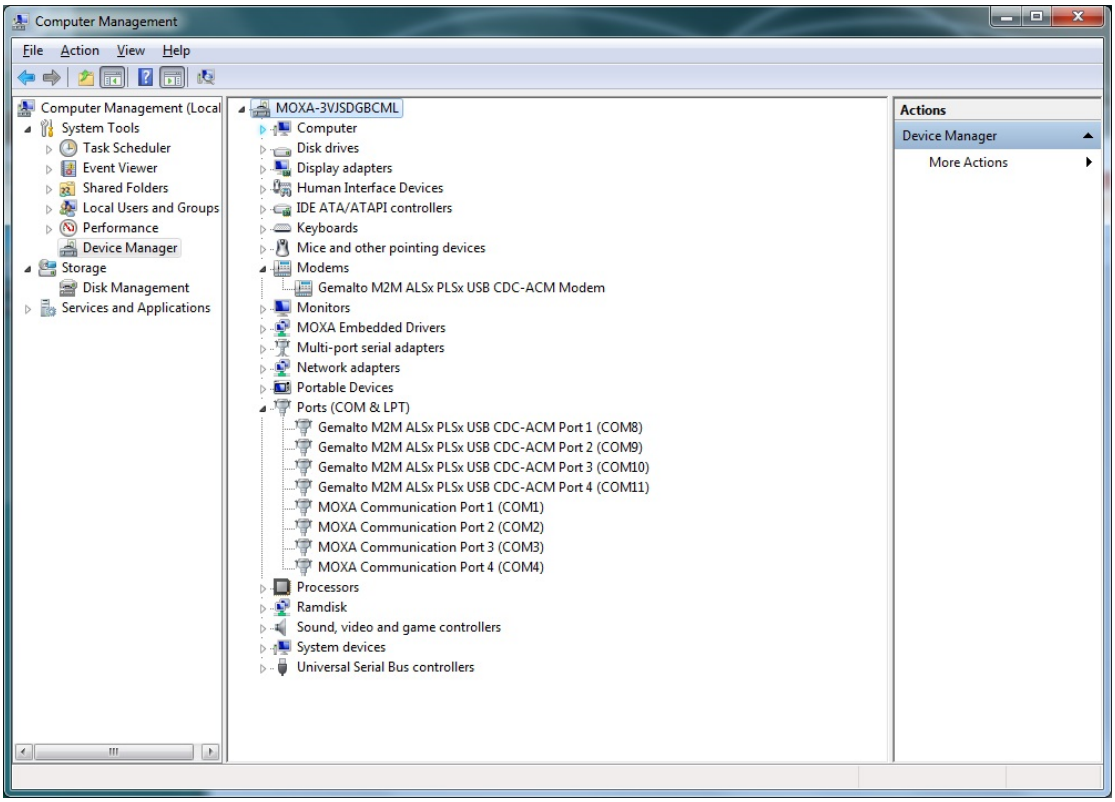
6. Click **OK** to complete installation of the LTE driver and utility.



7. Click **Close** to exit from the installation wizard.



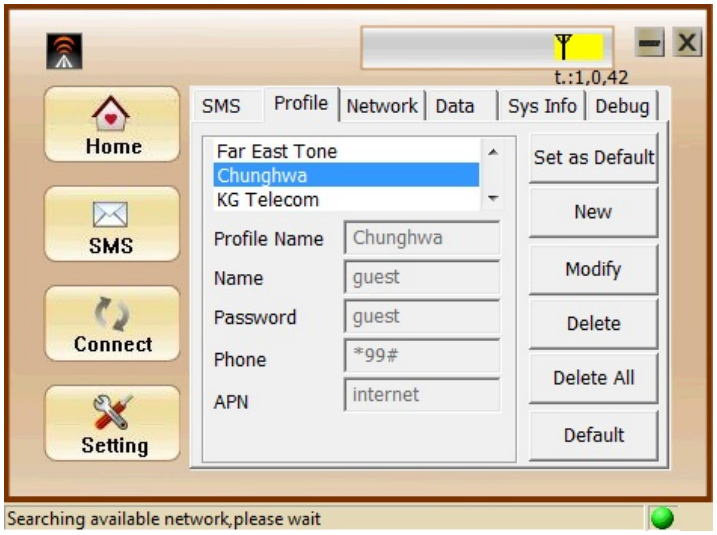
8. Open the Windows **Device Manager** again to confirm that the module is installed.



LTE Module Configuration

1. Insert the LTE SIM card into one of the SIM card holders.
2. Attach the cellular antenna to the LTE module connectors.
   (main and div connectors)
3. Run the Connection Manager.
4. Enter the PIN code to unlock the SIM card.

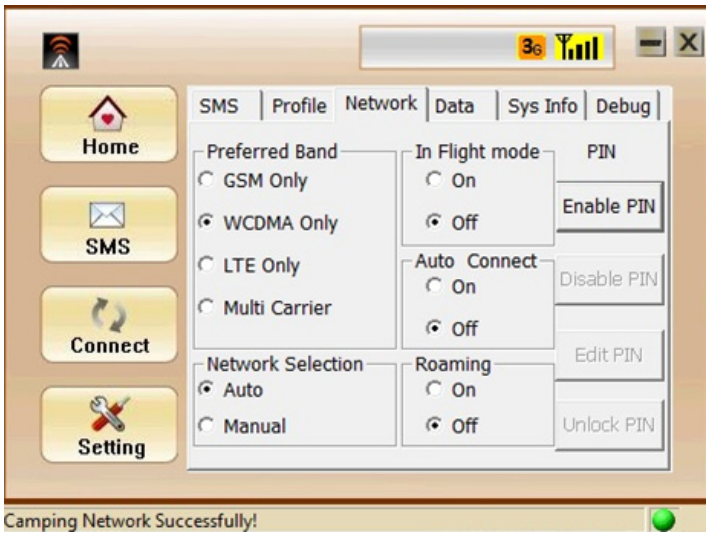
5. Click **Setting** on the left panel. Wait till the Connection Manager searches for all available networks and then select your service provider from the list.



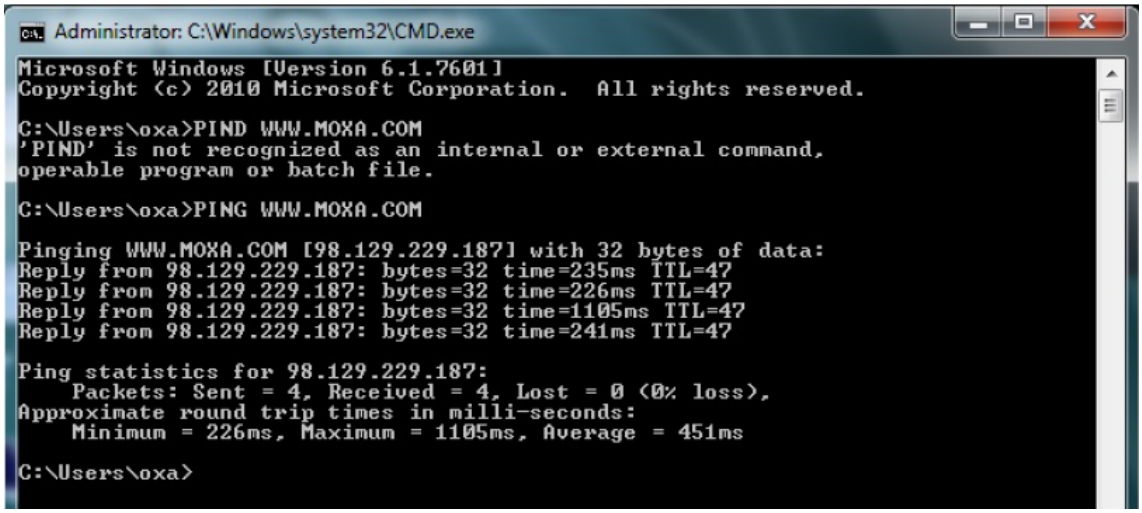
6. Click **Connect** on the left panel. After a connection is successfully established, the **Ready** message is shown in the status bar and the Tx/ Rx throughput values are shown on the main panel.

7. (Optional) Network setting for LTE mini card.

   The default **Preferred Band** setting in the **Network** tab of the Connection Manager is **WCDMA Only**, which can only be used for 3G connections. If you want to use an LTE mini card, you must select either **LTE only** or **Multi Carrier** as the **Preferred Band** before you try to establish a connection**.**

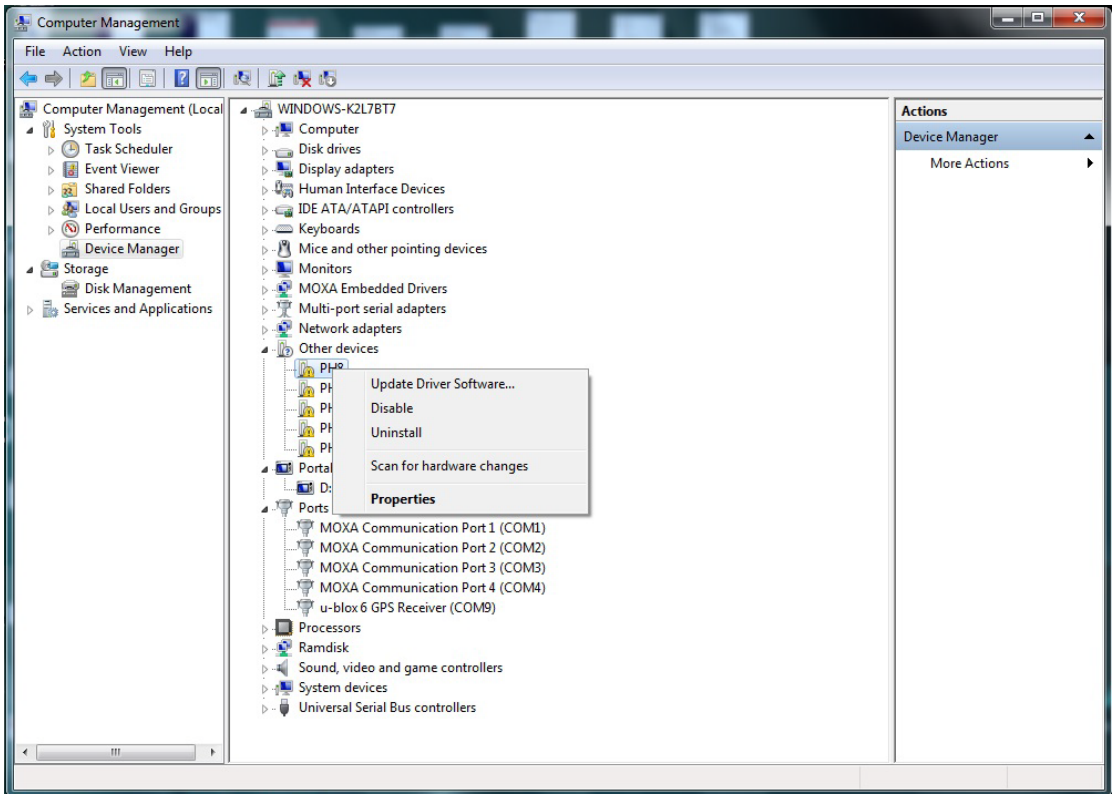


You can also verify the cellular connection once the connection is established using the ping command as shown below:

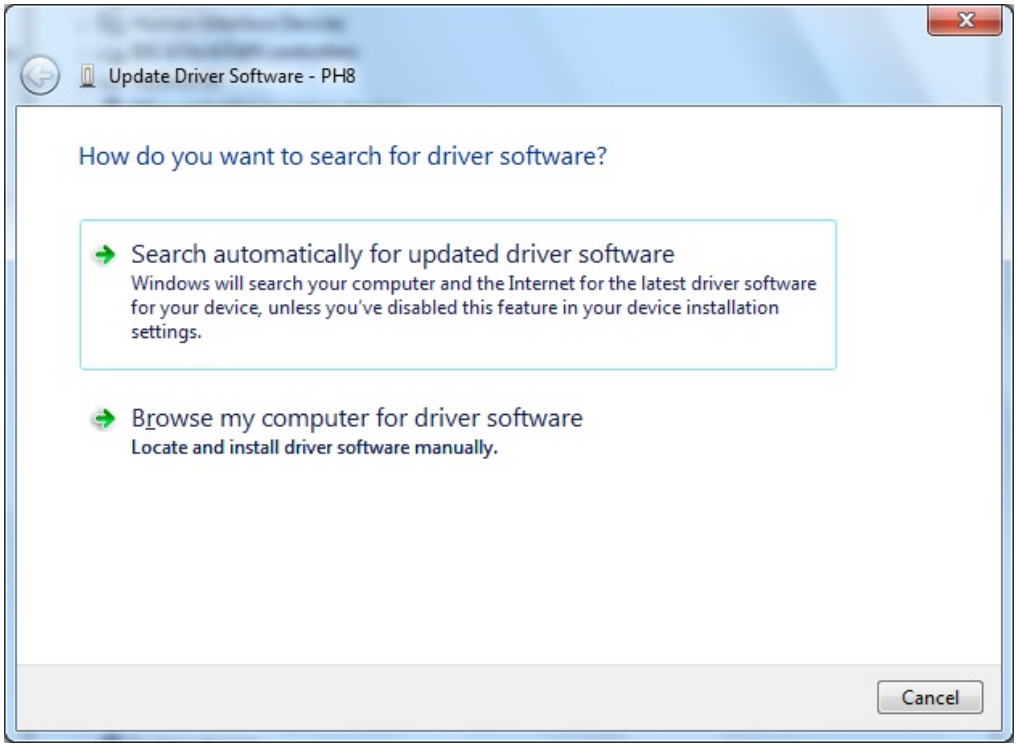


3G Module Driver Installation

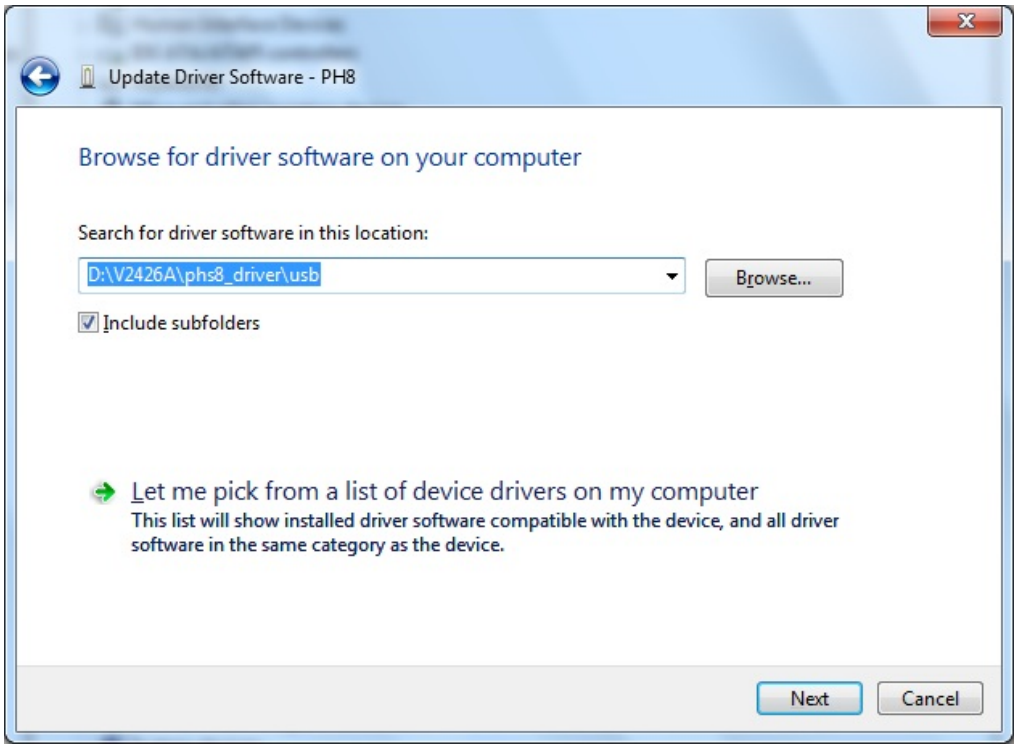
1. Open **Windows Device Manager** and right click to "PHS8" and select Update Driver Software.



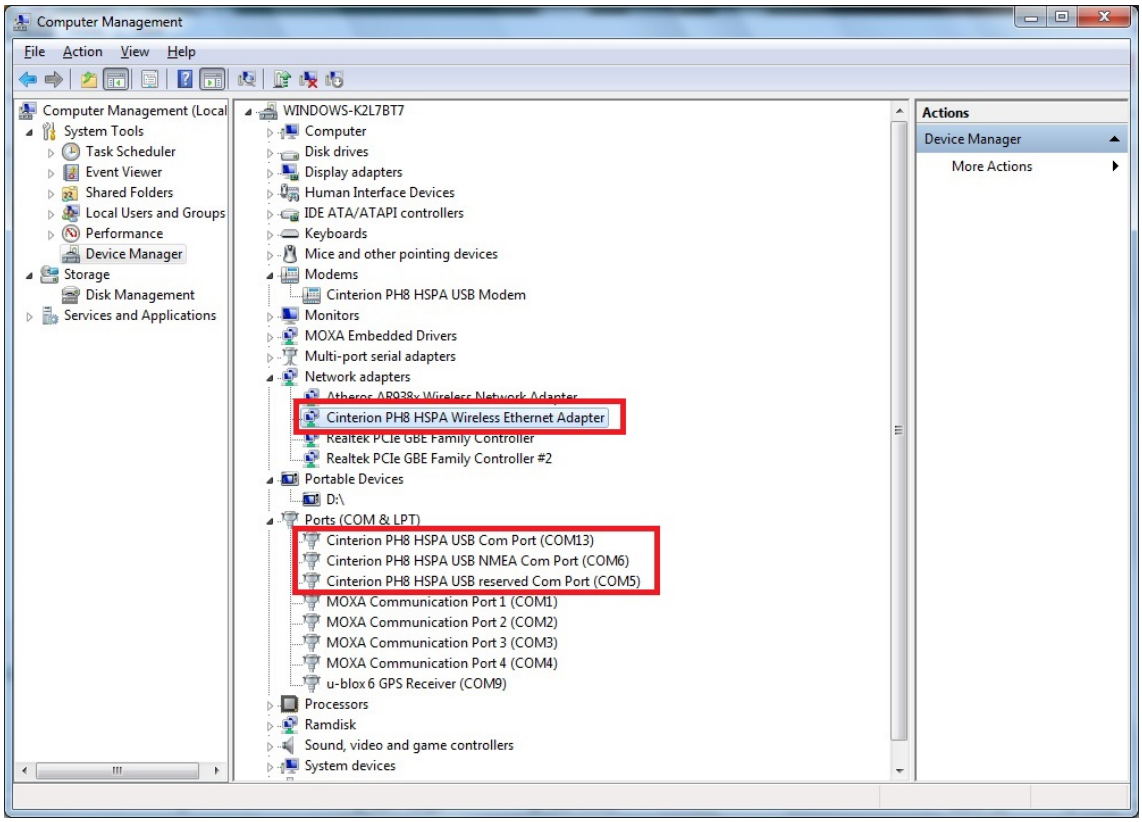
2. Click "Browse my computer for driver software"



3. Click **Install**.

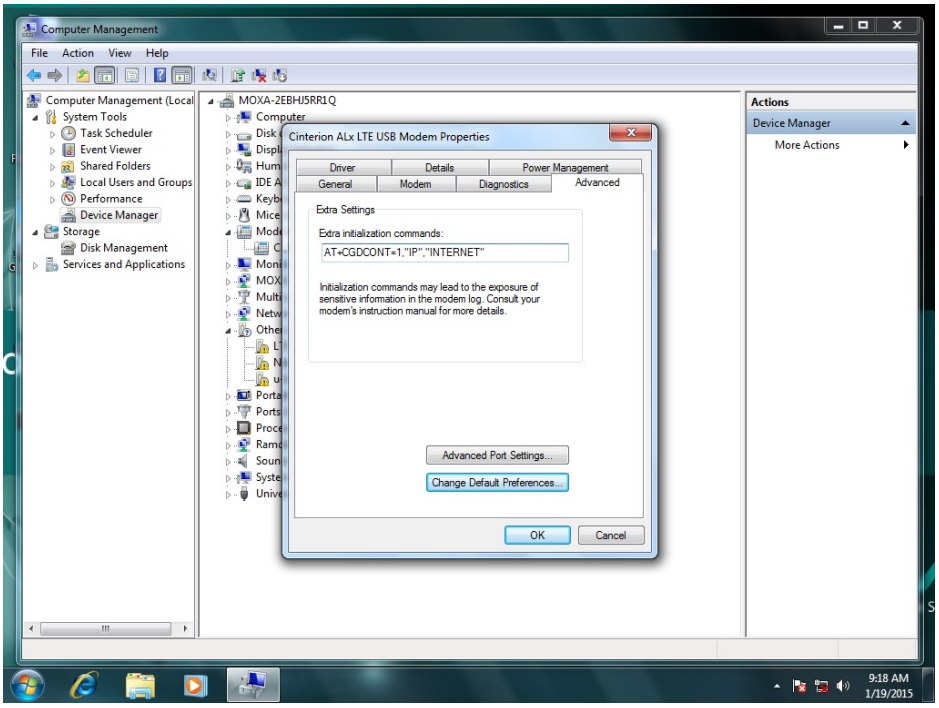


4. Redo step 1 to 3 to other "PHS8", cellular module and GPS module driver will both install completely in this section.

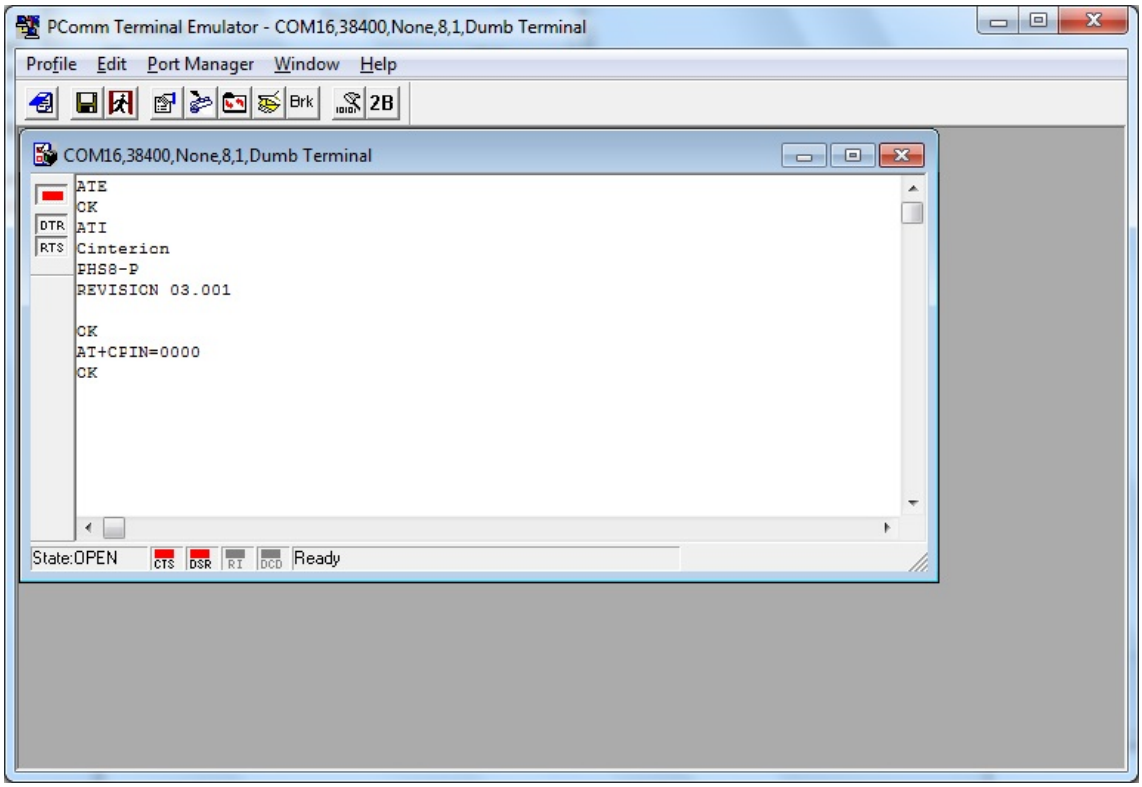


3G Module Configuration

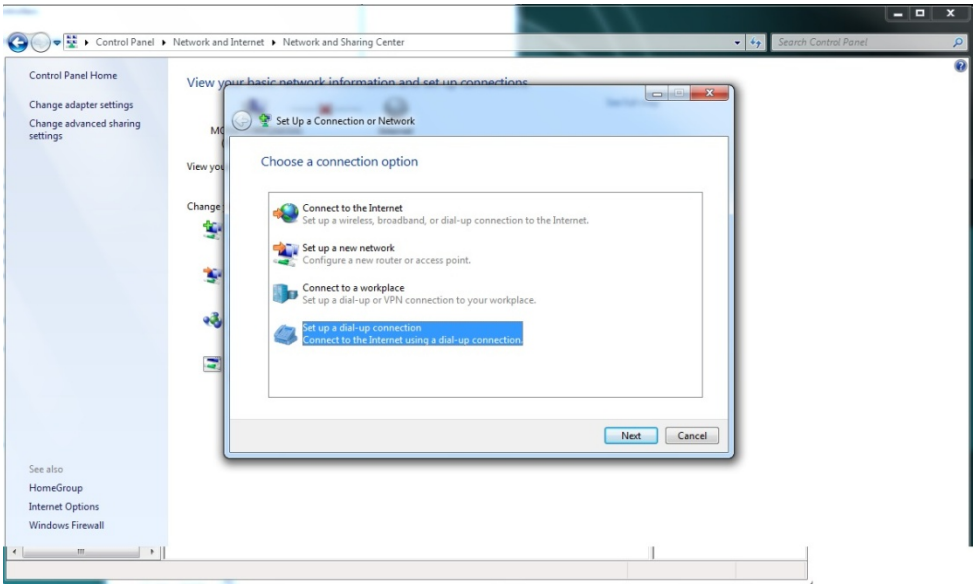
1. After installing cellular module driver, Open the **Windows Device Manager** and right click "Cinterion PHS8 HSPA USB Modem" port and enter AT command (format: AT+CGDCONT=profile, PDP type, APN).
   For example, if your ISP is Chunghwa telecom, enter " AT+CGDCONT=1, "IP", "INTERNET" "



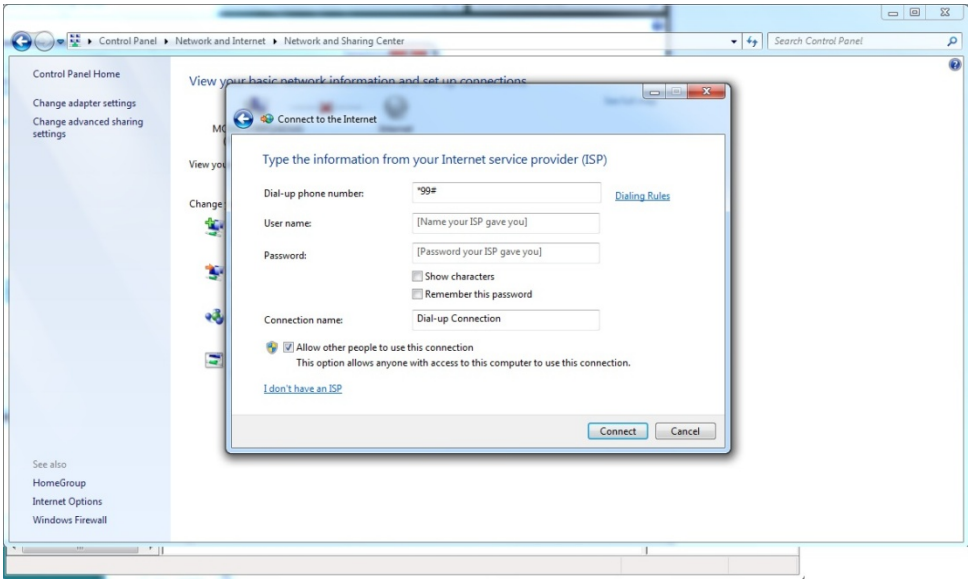
2. Start **Mxterm** utility and open these ports, then enter the **AT+CPIN=your pin code** command in the Cinterion ALx LTE USB Modem terminal.



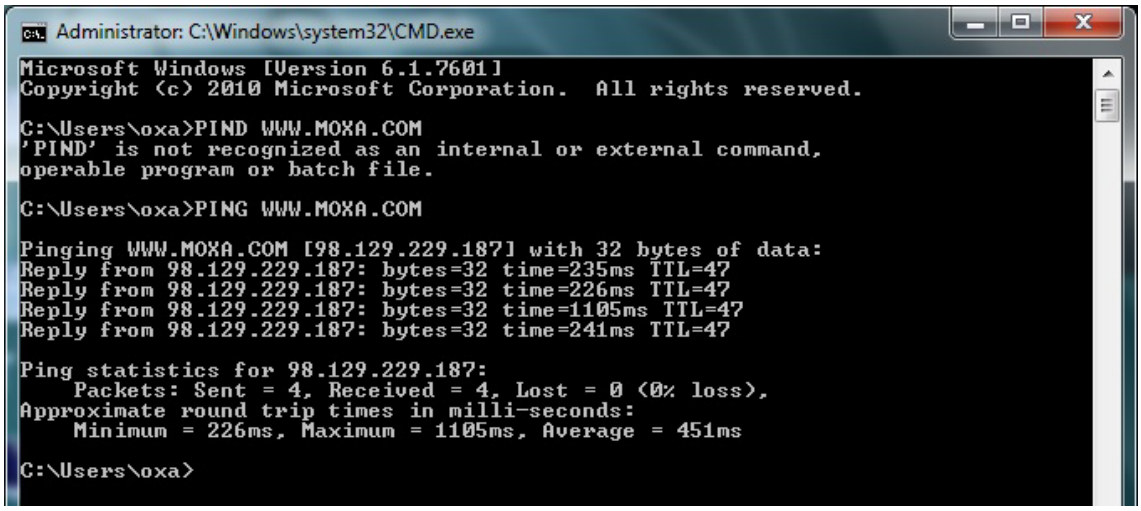
3. From control panel/ Network and Internet/ Network and Sharing center, select "Set a new connection or network" and choose "Set Up a dial-up connection"



4. In Connect to the internet page, enter the phone number which ISP gave you.
   For example, if your ISP is Chunghwa telecom, enter "*99#" in Dial-up phone number



5. You can verify the cellular connection by pinging the interface once the connection is established.

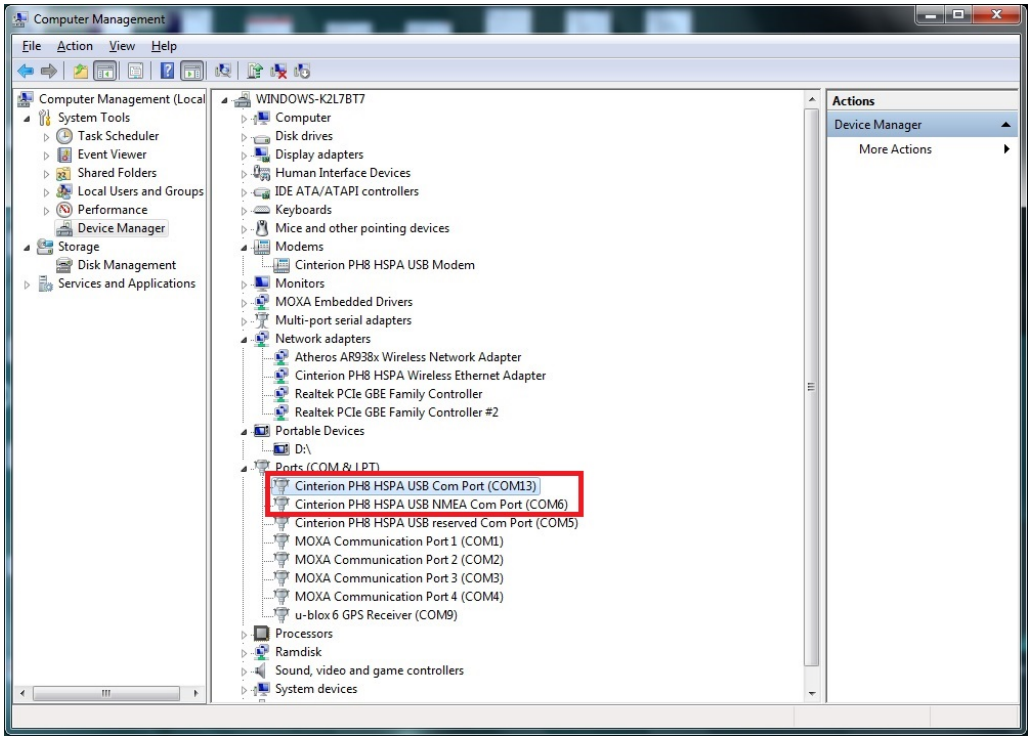


3G-GPS Module Driver Installation

Please refer to the section in Chapter 4: **3G Module Driver Installation**.

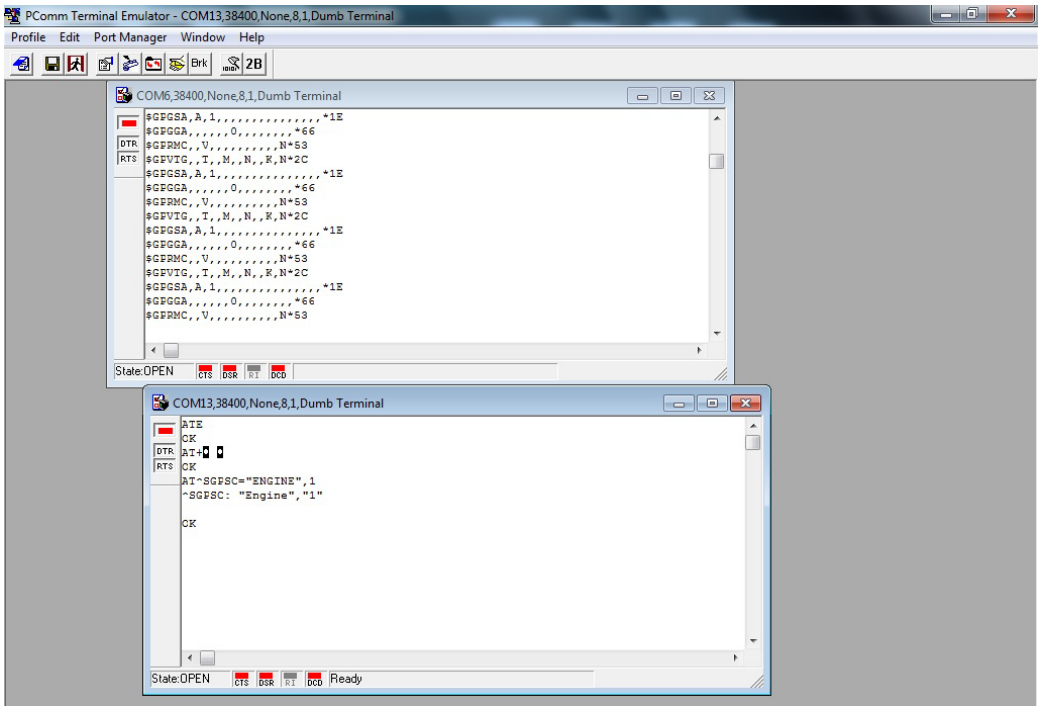
3G-GPS Module Driver Configuration

1. Start the **Device Manager** and check the "Cinterion PHS8 HSPA USB Com port" "Cinterion PHS8 HSPA USB NMEA Com port" and COM port.



2. Start **Mxterm** utility and open these ports, then enter the **AT^SGPSC="ENGINE","1"** command in the "Cinterion PHS8 HSPA USB Com port".

   You can then view the **information** returned by the GPS in "Cinterion PHS8 HSPA USB NMEA Com port" and verify that the device is working properly.



5

Software Utility

In this chapter we discuss installation and usage of software utility.

The following topics are covered in this chapter:

- **MxSerialInterface for the EPM-3032**
  - Overview
  - Installing MxSerialInterface for the EPM-3032
  - Configuring UART Mode
  - Uninstalling MxSerialInterface for the EPM-3032

MxSerialInterface for the EPM-3032

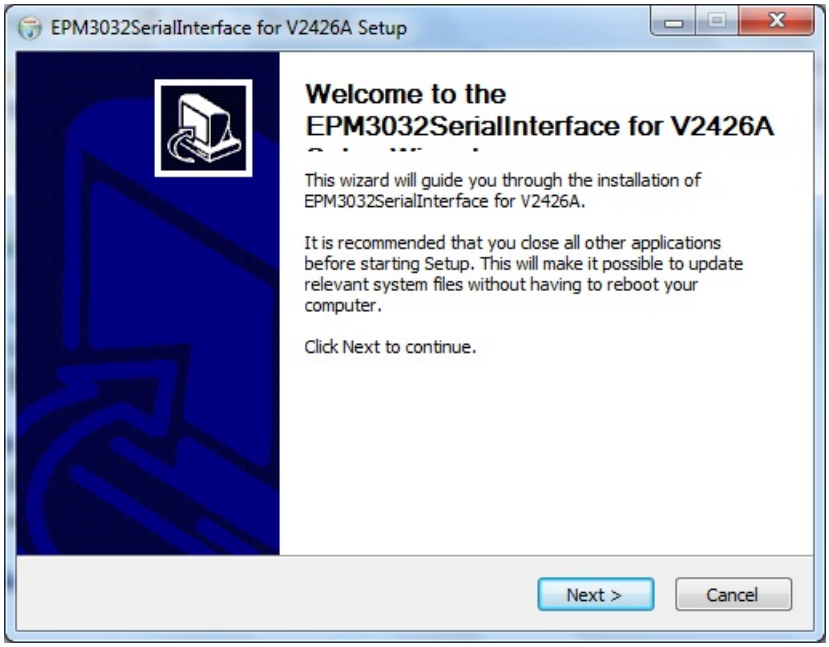
Overview

MxSerialInterface is a utility that provides a friendly user interface for users to set and save each UART mode.

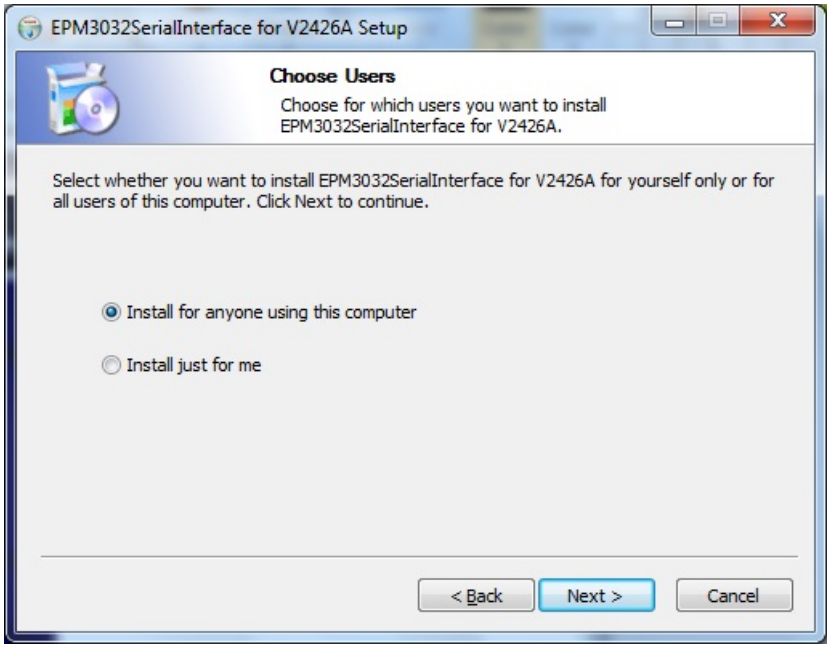
Installing MxSerialInterface for the EPM-3032

After installing the EPM-3032 expansion module, you need to install MxSerialInterface to set and save the EPM-3032's UART mode.

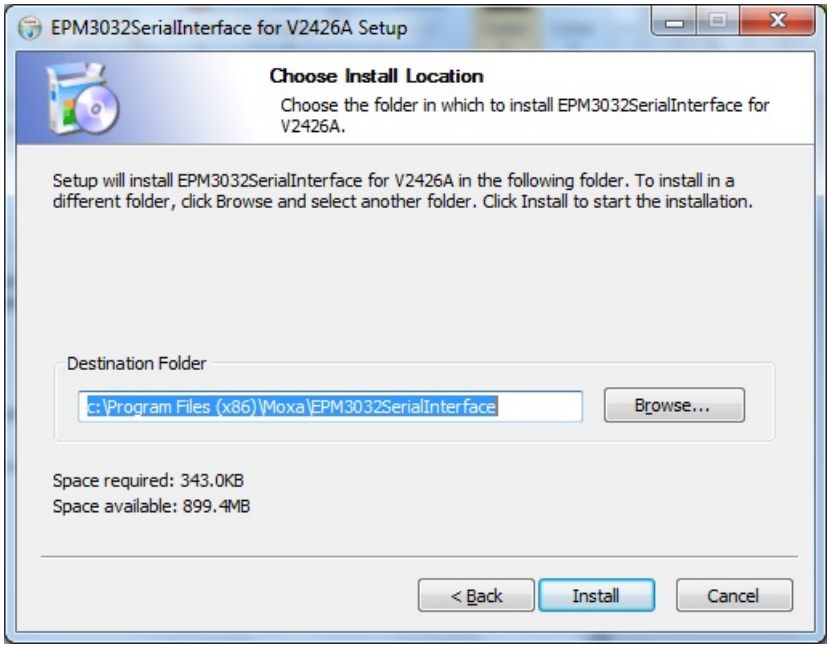
1. Locate **EPM3032SerialInterface_V2426A_1.0_x86_Setup.exe** on your software CD and execute it to install the utility. When the welcome screen opens, click **Next**.



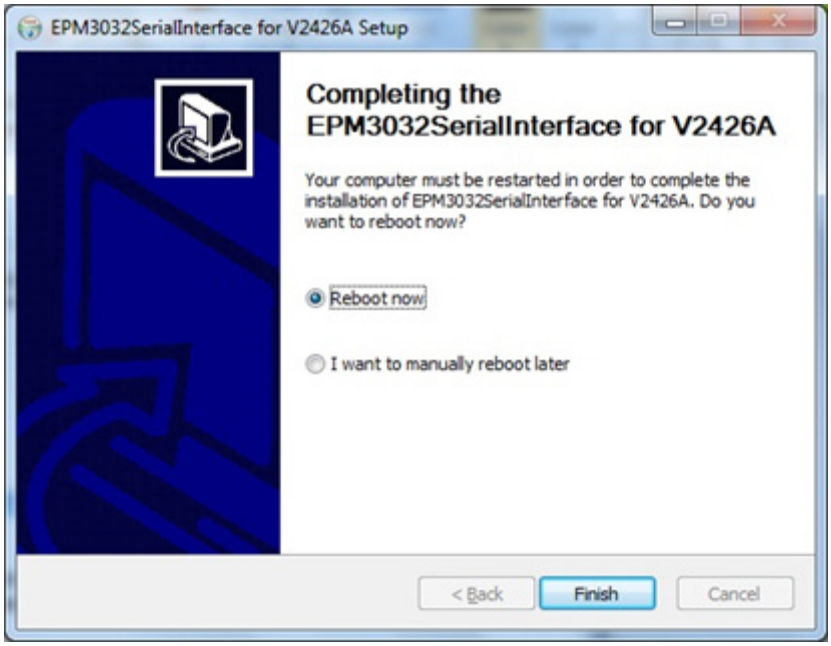
2. Click **Next** to install using default settings.



3. Click **Install** to start the installation.



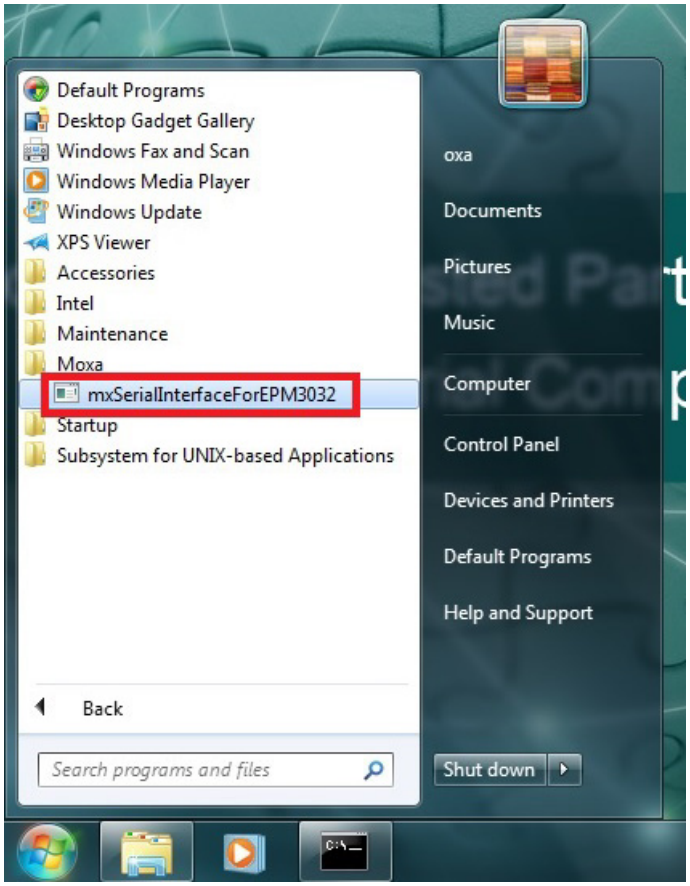
4. Click **Finish** to reboot the computer to complete the installation.



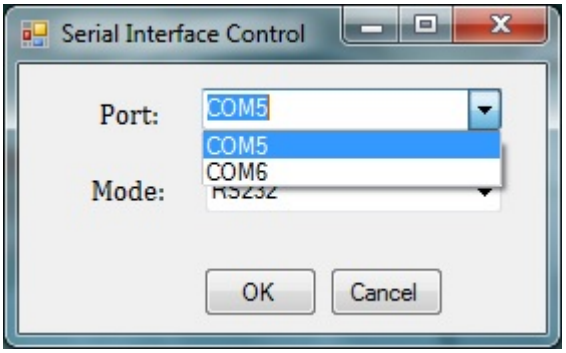
Configuring UART Mode

Follow these steps to configure UART mode.

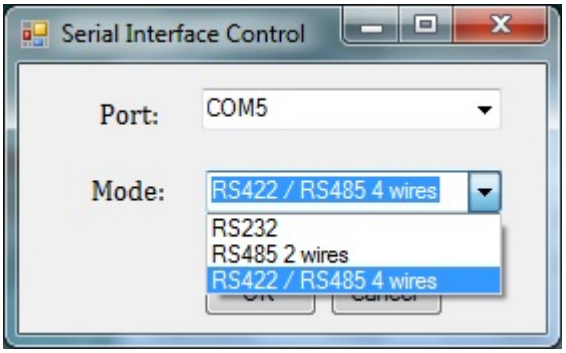
1. Click the **MxSerialInterfaceForEPM3032** shortcut.



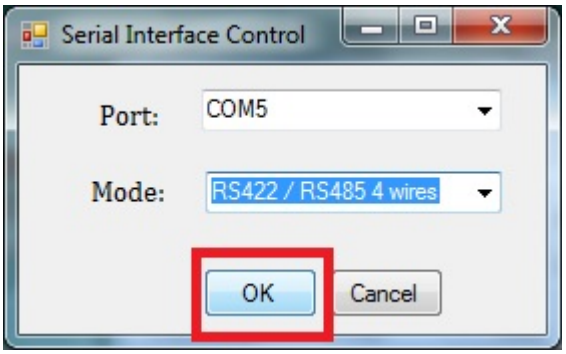
2. Select the port number that you want to use.



3. Select the mode that you want to use.

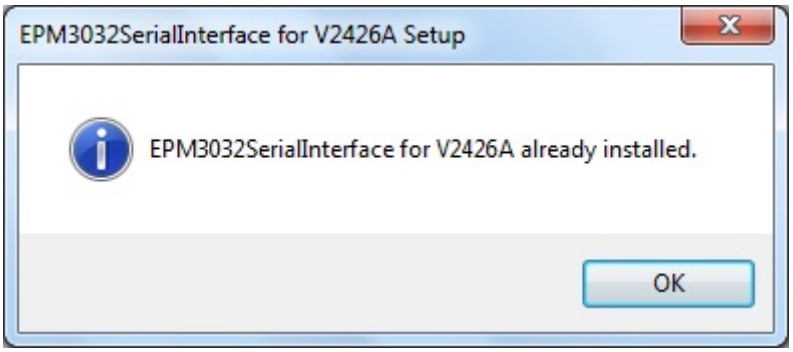


4. Click **OK** to complete the port configuration.

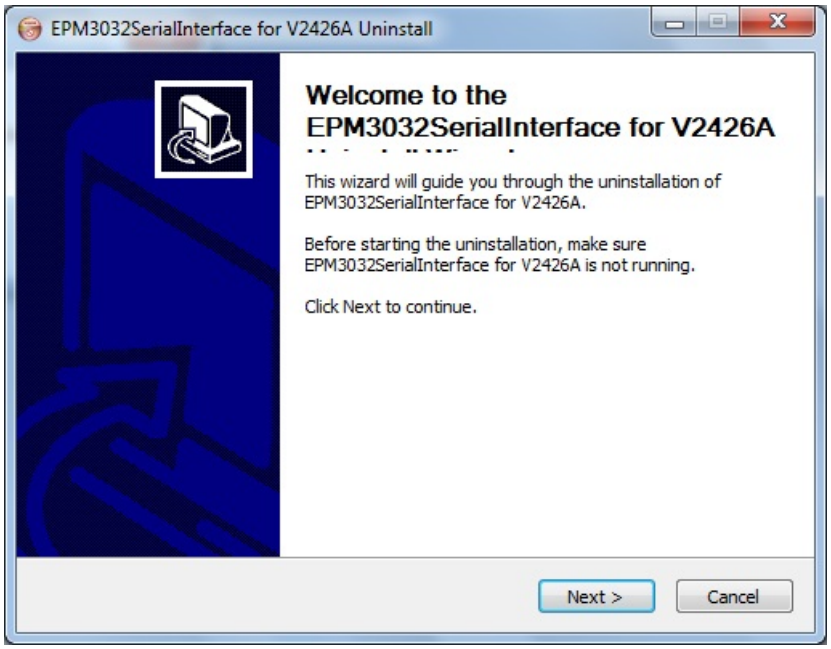


Uninstalling MxSerialInterface for the EPM-3032

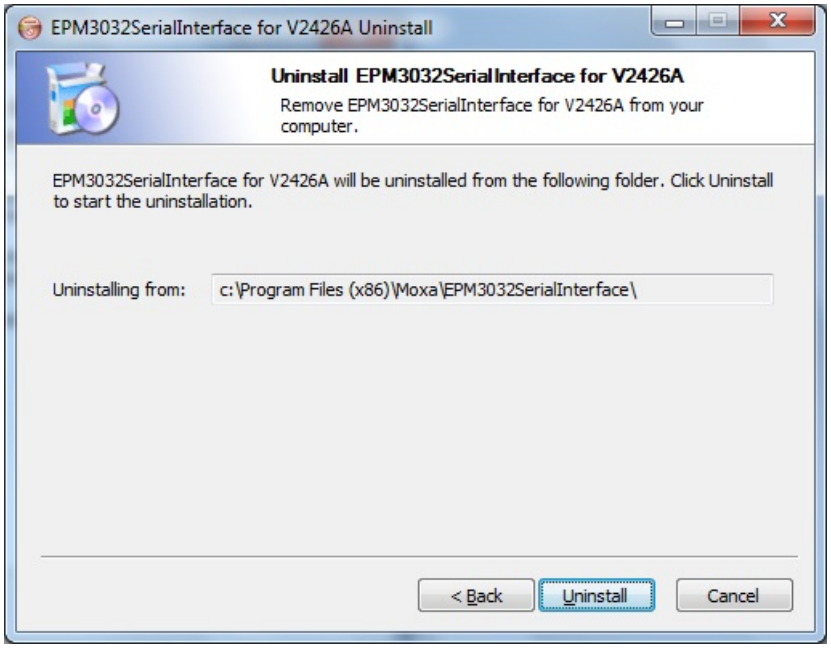
1. Locate **EPM3032SerialInterface_V2426A_1.0_x86_Setup.exe** on your software CD and execute it to install the utility. Click **OK**.



2. Click **Next**.



3. Click **Uninstall**.



4. Click **Finish** to reboot the computer to complete uninstalling the software.