Programmer's Guide for Moxa's Windows CE Embedded Computers

Third Edition, March 2008

www.moxa.com/product

Programmer's Guide for Moxa's Windows CE Embedded Computers

Copyright Notice

Trademarks

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 made periodically to the information in this manual 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 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 China (Beijing office):
Tel: +86-10-6872-3959/60/61
Fax: +86-10-6872-3958

Table of Contents

1

Installing Development Tools

A number of well-known Integrated Development Environments (IDE) tools can be used to ease the development of applications on Moxa Windows® CE embedded computers. Choose the tools based on the application language that you plan to use, and then install the tools on your development workstation. C++ developers can choose Visual Studio 2005 or eMbedded Visual C++ 4.0 (eVc 4.0). eMbedded Visual C++ 4.0 can be downloaded for free from the Microsoft MSDN website, but Windows CE 6.0 does not support eMbedded Visual C++ 4.0.

In this chapter, we present the steps you should follow to install development tools for Windows® Embedded Application Development.

The following topics are covered in this chapter:

- **Visual Studio 2005**
  - Installing Visual Studio 2005
  - Uninstalling Microsoft .Net Compact Framework 2.0
  - Installing the .Net Compact Framework 2.0 with Service Pack
- **eMbedded Visual C++ (eVC) 4.0**
- **Moxa Windows CE C/C++ SDK**

Visual Studio 2005

Microsoft® Visual Studio 2005 is a complete set of development tools for building ASP.NET Web applications, XML web services, desktop applications, and mobile applications. The same IDE is used by Visual Basic, Visual C++ and Visual J#, allowing them to share tools and facilitating the creation of mixed-language solutions.

Visual Studio 2005 Setup will install .Net Compact Framework 2.0 on your development station. You will also need to install Service Pack 1 for .Net Compact Framework 2.0 in order to match the version used by the Moxa WinCE embedded computer. When installing the service pack, you will be prompted to remove any older versions of .Net Compact Framework 2.0 that are already installed on your computer.

If you do not install the latest service pack for .Net Compact Framework 2.0, you may encounter problems when using Visual Studio 2005's on-line debugging function.

Installing Visual Studio 2005

VB.NET/C# developers should follow this sequence when installing Visual Studio 2005:

1. Install Visual Studio 2005
2. Uninstall Microsoft .NET CF 2.0
3. Install Microsoft .NET CF 2.0 Service Pack

C/C++ developers should follow this sequence when installing Visual Studio 2005:

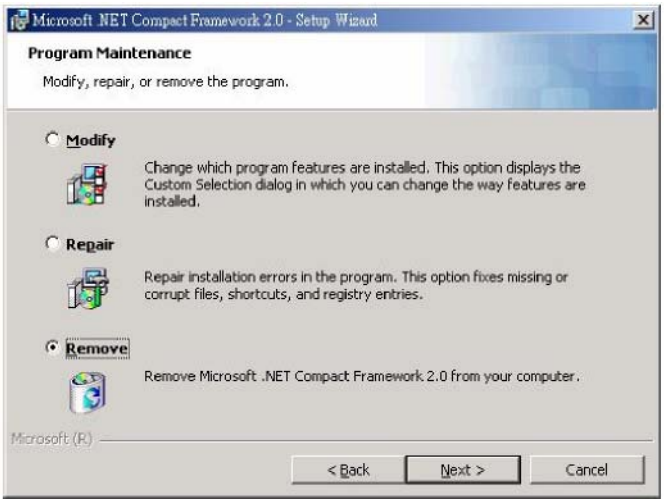
1. Install Visual Studio 2005
2. Install Moxa Windows® CE C/C++ SDK

For the details on system requirements and installation procedures, please refer to Microsoft's documentation or to MSDN.

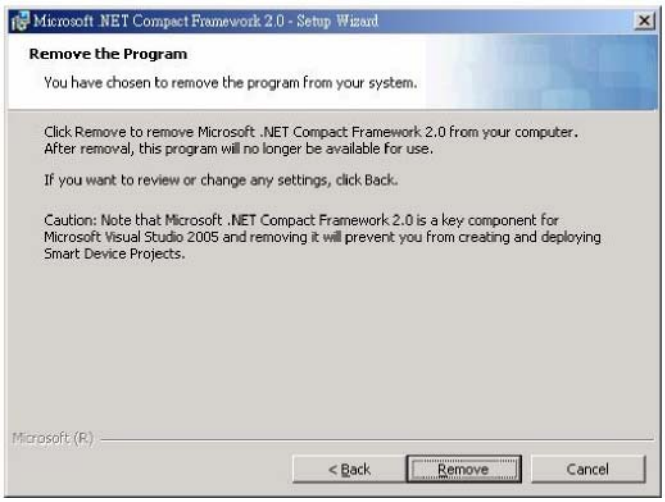
Uninstalling Microsoft .Net Compact Framework 2.0

Open the Control Panel and locate .Net Compact Framework 2.0.

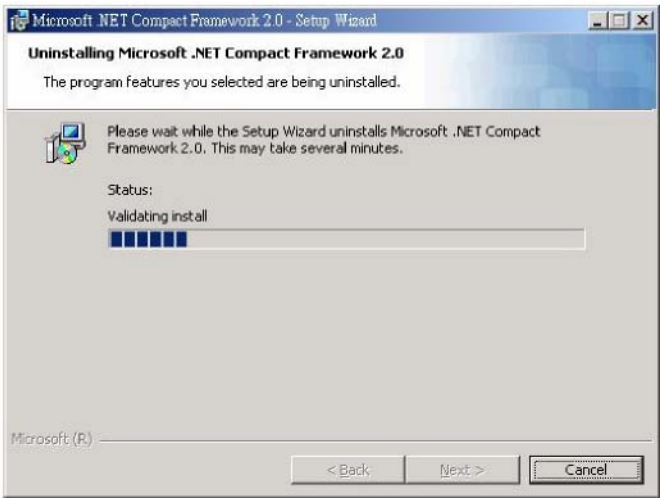
1. Select **Remove** and click **Next** to proceed.



2. Click **Remove** to confirm that you would like to remove the software.



3. Wait patiently while the software is uninstalled from your computer.



4. Click **Finish** to exit the wizard.

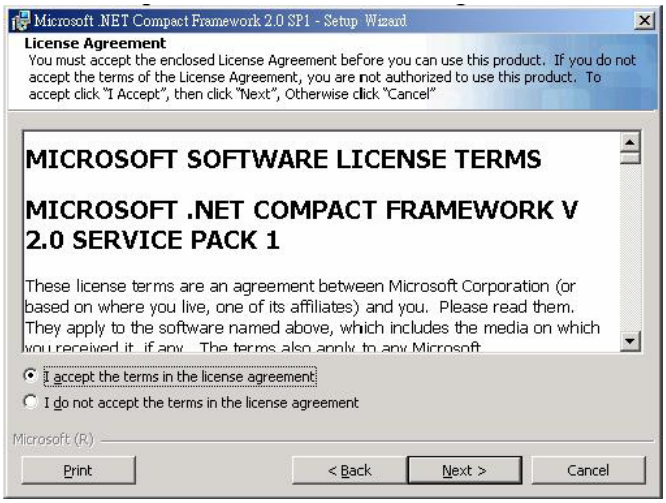
You may now install Service Pack 1 for.Net Compact Framework 2.0.

Installing the .Net Compact Framework 2.0 with Service Pack

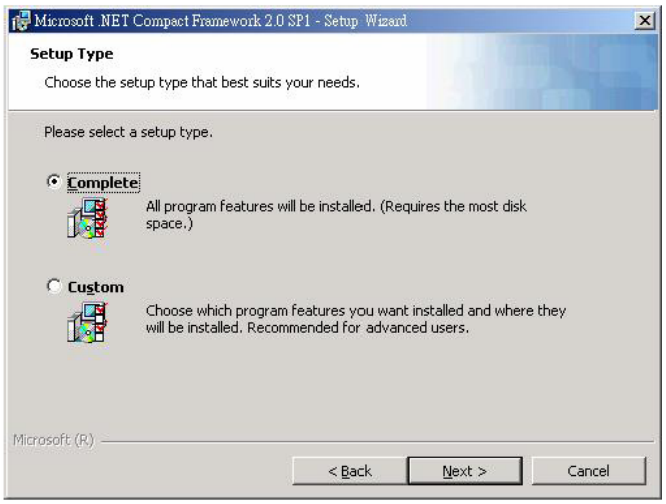
1. Wait while the Windows Installer prepares to install the software.

2. Click **Next** to proceed with the installation.

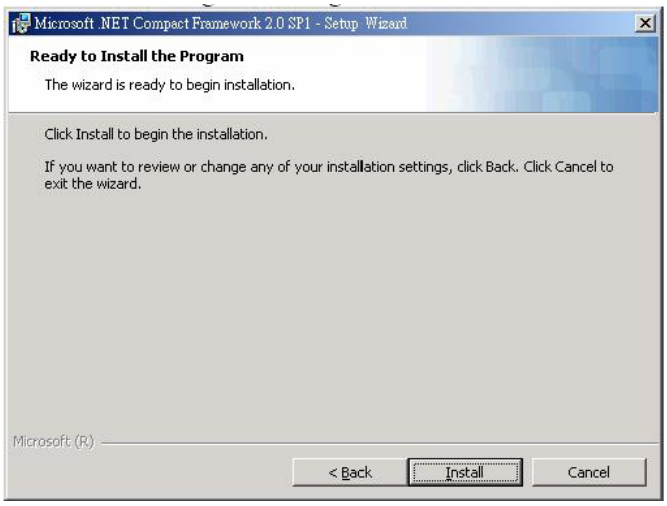
3. Select **I accept the terms in the license agreement** and click **Next** to proceed.



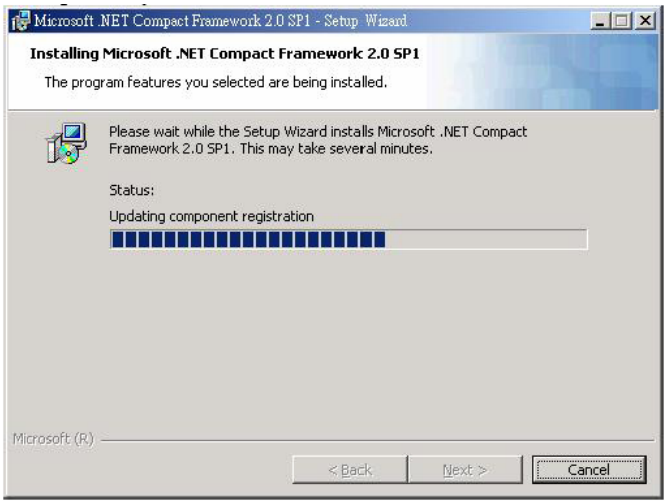
4. Select **Complete** and click **Next** to proceed.



5. Click **Install** to begin installing the software.



6. Wait patiently while the software is installed.



7. Click **Finish** to complete the installation procedure.

eMbedded Visual C++ (eVC) 4.0

The eVC 4.0 tools can be downloaded for free from the MSDN's eMbedded Visual Tools Download Page. Install the eVC 4.0 tools, and then install the service pack in the following order:

- Install eMbedded Visual C++ 4.0 (230 MB)
- Install Service Pack 4 for eVC 4.0 (68 MB)
- Install Moxa Windows® CE C/C++ SDKs

NOTE:	Only Windows CE 5.0 supports eMbedded Visual C++ 4.0; for models shipped with Windows CE 6.0, please use Visual Studio 2005.

Moxa Windows CE C/C++ SDK

After installing eVC 4.0 or Visual Studio 2005 on the development workstation, C++ developers will need to install the C/C++ SDK from Moxa. The SDK file (e.g., ***UC7400CESDK1_2.msi***) is available on the CD provided by Moxa, or from the Moxa download center at http://web4.moxa.com/support/download_center.asp. To download the file from Moxa's website, once the "Download Center" web page opens, select your product from the "Select a Product" drop-down list under "Driver & Software Downloads" and then click **Go**. Next, scroll down towards the bottom of the page and click on the SDK link (e.g., **SDK_1.2.zip**) to download the file.

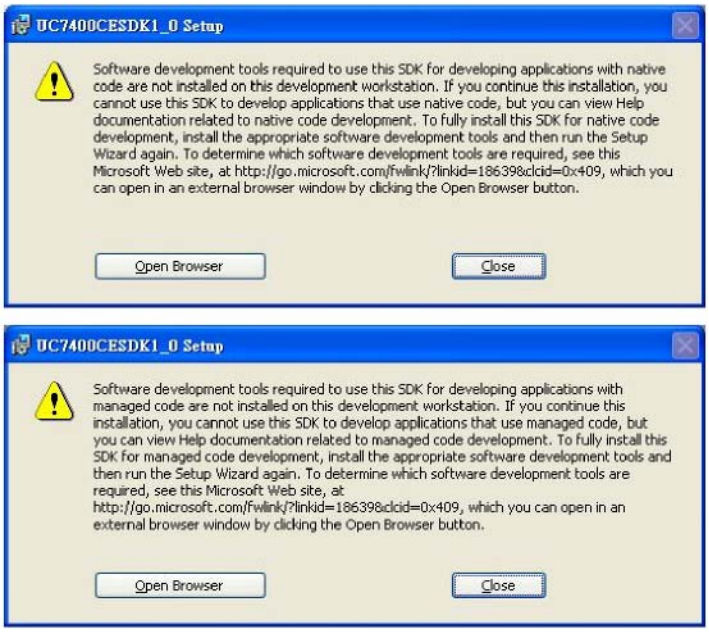
1. Double click the MSI file to open it. A Setup Wizard will appear to start the installation process. Click **Next** to proceed.
2. If an error message appears, choose **Close** to continue the installation process. This message indicates that the appropriate versions of the application development tools required by the SDK are not installed. Ignore this message if you do not need these tools. Otherwise, cancel the process and download appropriate files from the Microsoft® site. For example, if a dialog such as the following pops up, download the following two files:

   Microsoft® .NET Framework Version 2.0 Redistributable Package (x86) (25 MB)

http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

NET Framework 2.0 Software Development Kit (SDK) (x86) (354 MB)

http://www.microsoft.com/downloads/details.aspx?FamilyID=FE6F2099-B7B4-4F47-A244-C96D69C35DEC&displaylang=en

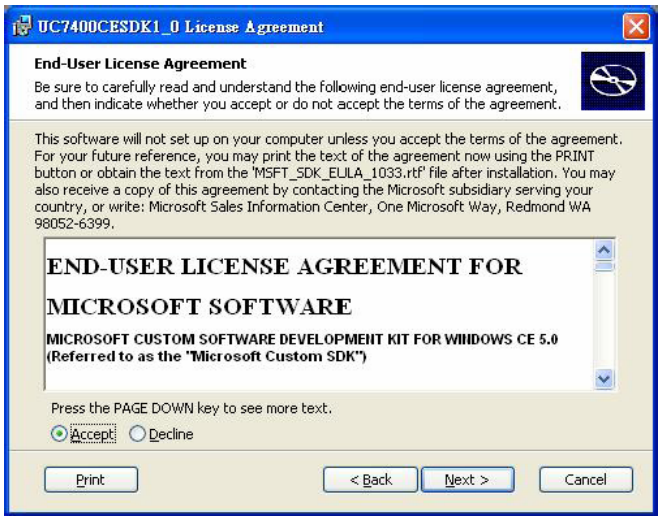


3. Assuming that you have saved these two files as **dotnetfx** and **setup**, take the following steps to install the tools:

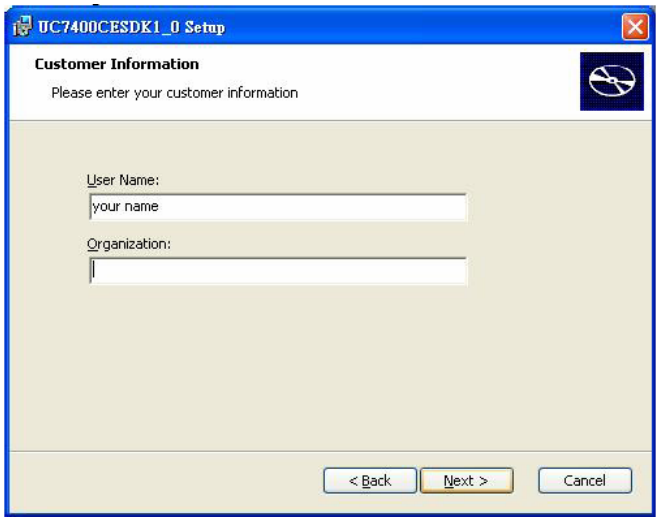
   - Execute the program **dotnetfx**.
   - Install .Net Framework by executing the program **setup**.

Next, reinstall the Moxa SDK.

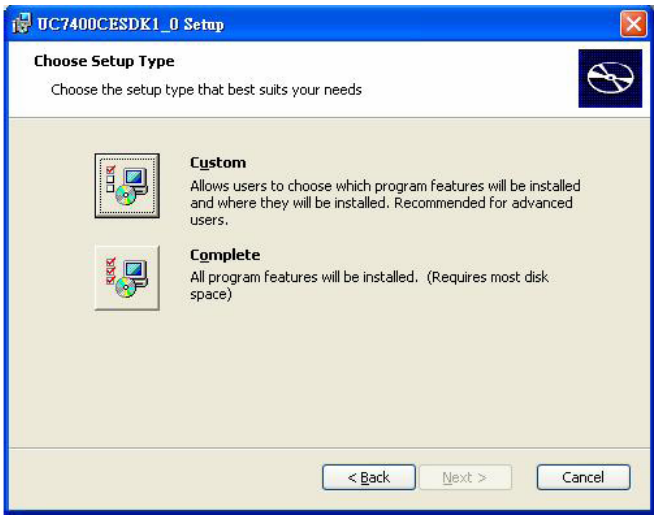
1. Read the "License Agreement" and if you accept the terms of the end-user license agreement (EULA), click the **Accept** radio button, and then click **Next**.



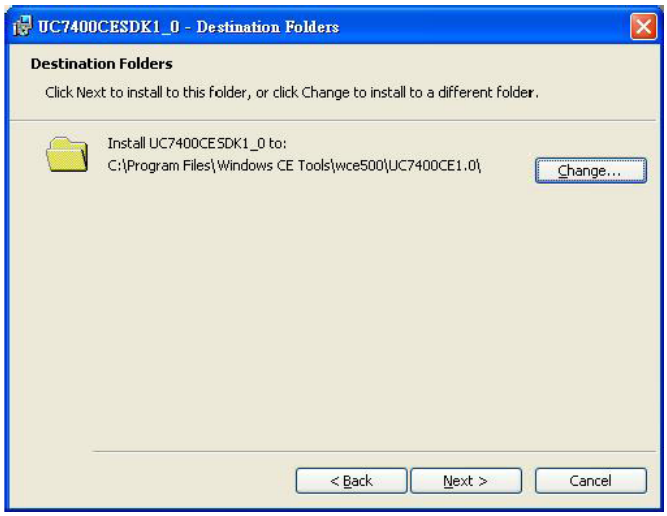
2. Type your name for **User Name**, and the name of your company for **Organization**. Click **Next** to proceed.



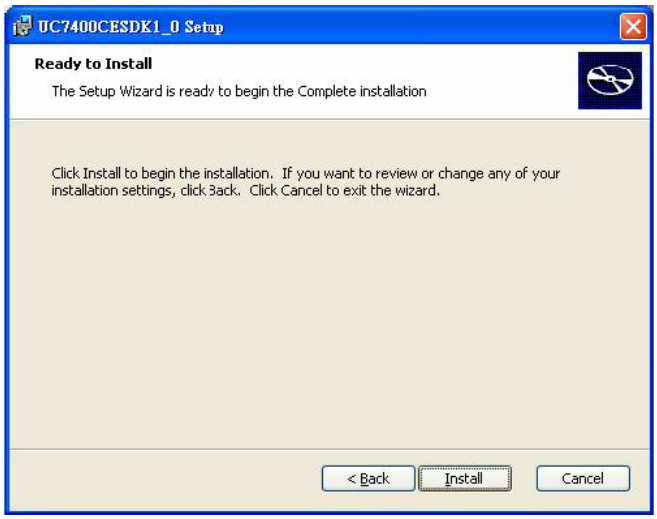
3. To install all functionality provided with the SDK, click **Complete** and then click **Next**.



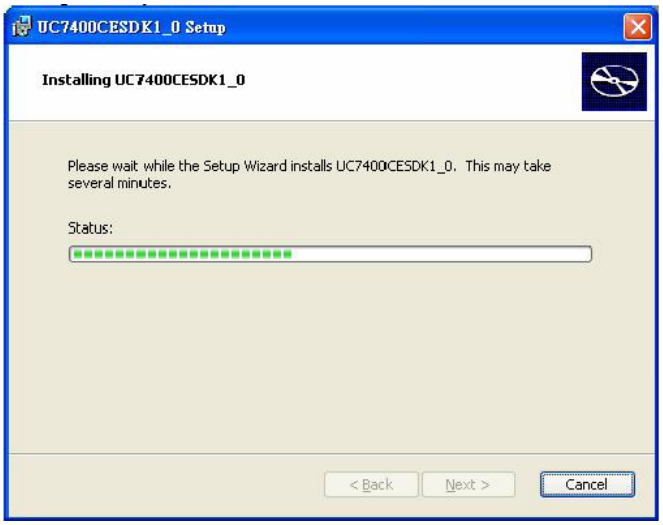
4. To change the name of the folder in which the SDK will reside, click **Change** and browse to the folder. Otherwise, click **Next** to proceed.



5. To install the SDK, click **Install**.



6. Wait patiently while the software is installed.



7. When the installation is complete, click **Finish** to close the installer.

2

Application Development

Application developers that are familiar with Windows® IDE tools and application programming interfaces (APIs) will appreciate the versatility of scripting languages that is found with Windows® CE and the .NET Compact Framework environment. The Moxa SDK enables developers to easily develop native code for Moxa embedded computers. In this chapter, we present examples using Visual Studio 2005 and eMbedded Visual C++ 4.0 in order to demonstrate the easy process of application development.

Both tools include on-line debugging for quicker and easier development. As with most PC-based development tools provided by Microsoft, the debugging methods follow a step-by-step approach.

The following topics are covered in this chapter:

- **Developing an Application with Visual Studio 2005**
- **Debugging an Application with Visual Studio 2005**
- **Developing an eMbedded Visual C++ 4.0 Application**
- **Debugging an eMbedded Visual C++ 4.0 Application**

Developing an Application with Visual Studio 2005

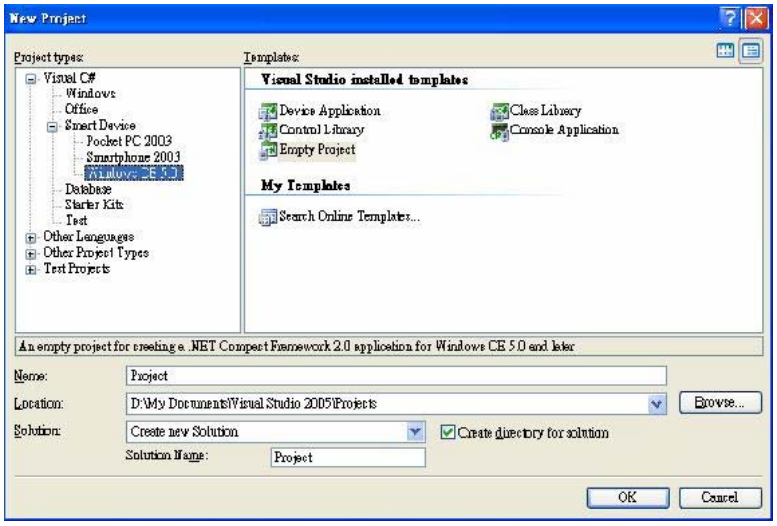
- Open Microsoft® Visual Studio .Net 2005.
- From the **File** menu, choose **New → Project.**
- Choose the **Project Type** and then select the **Smart Device Application** as the type of project.
- Fill in the project name and click **OK.**
- Choose **Windows CE** as the target platform.
- Select the desired project type and click **OK.**
- Write your application code.
- From the **Device** toolbar, choose **Windows CE.Net Device.**
- From the **Build** menu, choose **Build Project or Rebuild Project.**
- When you complete your application, upload it to the embedded computer.
- Log on to the embedded computer. At the console prompt, execute it directly if it is a C# file. Otherwise, execute it using the program **wcescript.exe** if it is a VB script or java script file.

Debugging an Application with Visual Studio 2005

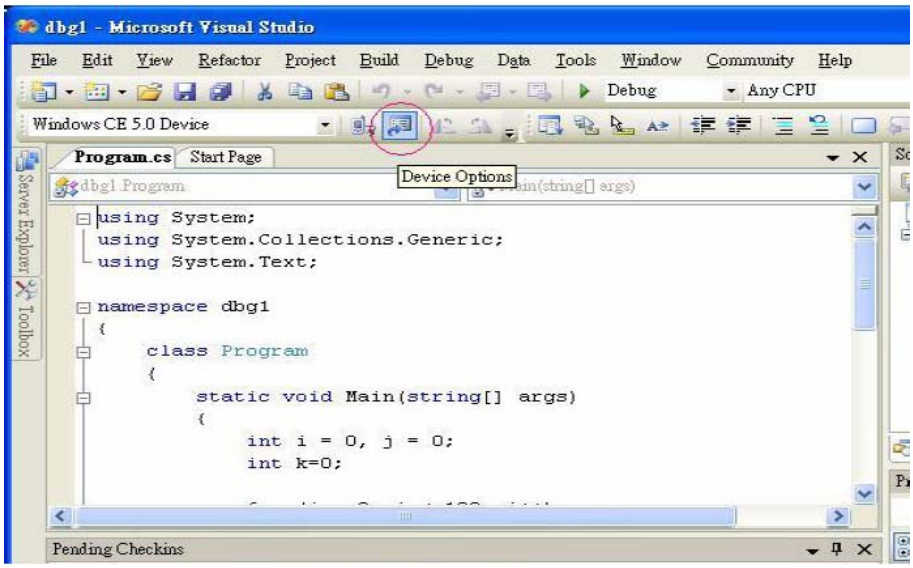
1. Before you use the remote tools, please make sure that no other ActiveSync device is connected to your workstation.
2. Find the six files listed below on your PC. The files are usually located in the folder **C:\Program Files\Common Files\Microsoft Shared\CoreCon\1.0\Target\wce400\<CPU architecture>**. Copy the files from that folder to your embedded computer.

   Clientshutdown.exe
   CMAccept.exe
   ConmanClient2.exe
   DeviceDMA.dll
   eDbgTL.dl
   TcpConnectionA.dll
3. Log onto the embedded computer and run *conmanclient2.exe* in the background by entering the following command:
   `\> start conmanclient2.exe`
4. Run *CMAccept.exe* in the background by entering the following command:
   `\> start CMAccept.exe`

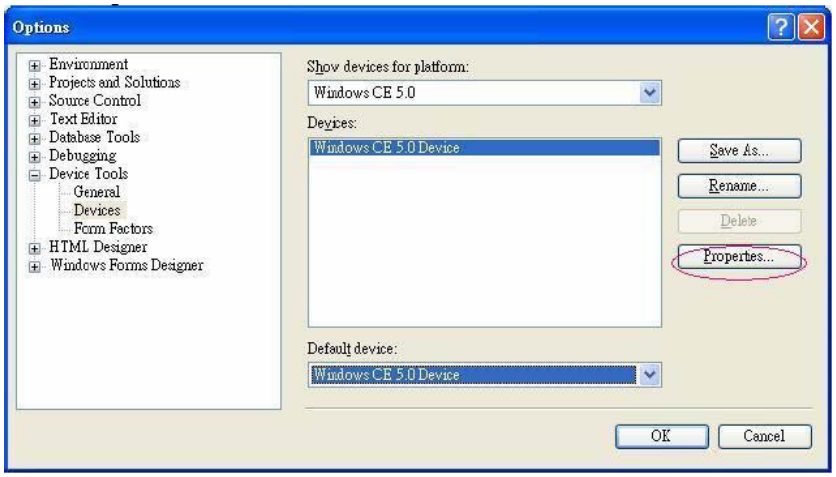
   NOTE: This program accepts a connection from Visual Studio 2005 and immediately ends. For a new connection, run the program again.
5. Open a Visual Studio 2005 Smart Device project. You may also start a new project as follows: **New Project → Select project type → Choose Smart Device → Select templates → Give a project name → Click OK**



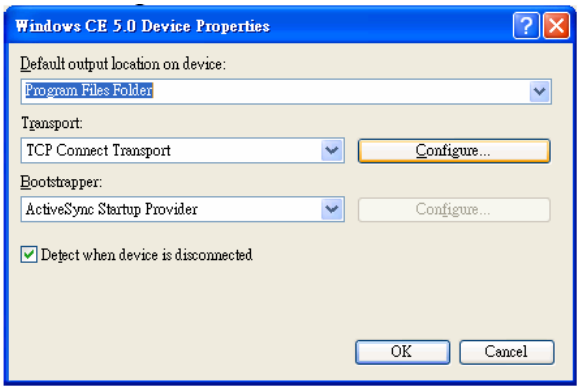
6. Click Device Option.



7. Click **Properties…**.

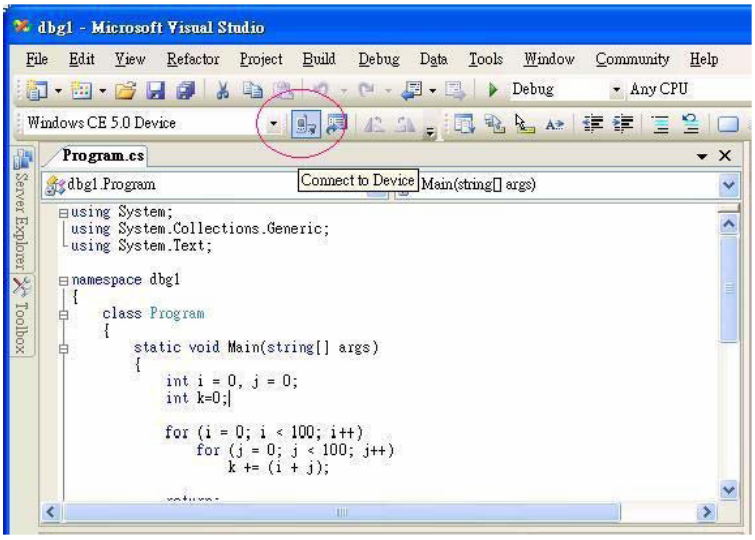


8. Click **Configure…**.



9. Click the **Use specific IP Address** radio button and enter the network address of the embedded computer. Click **OK** to continue.

10. Click the **Connect to Device** button.

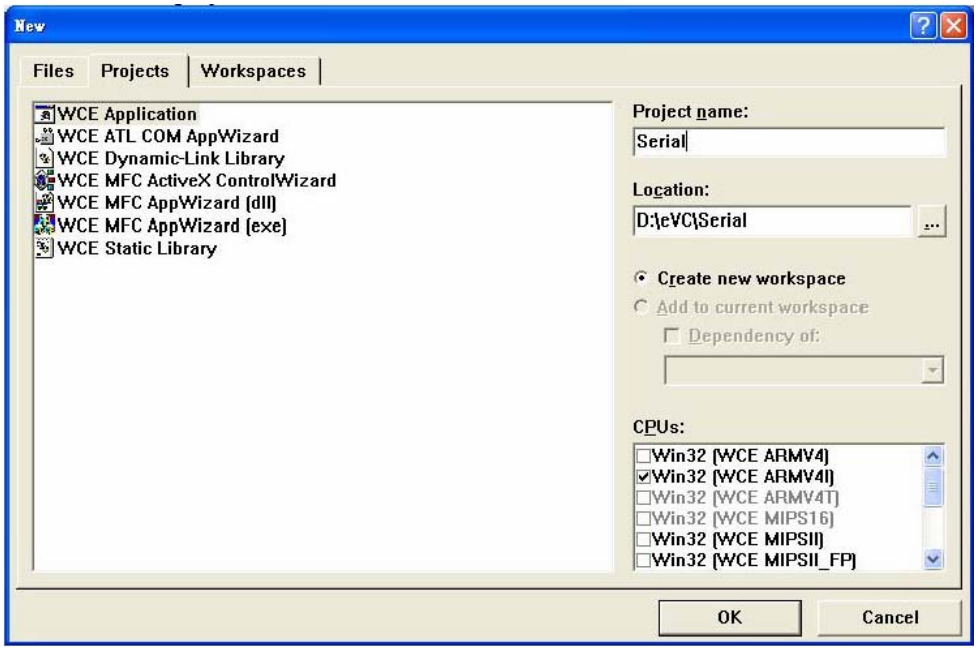


11. Wait for the connection to be made.

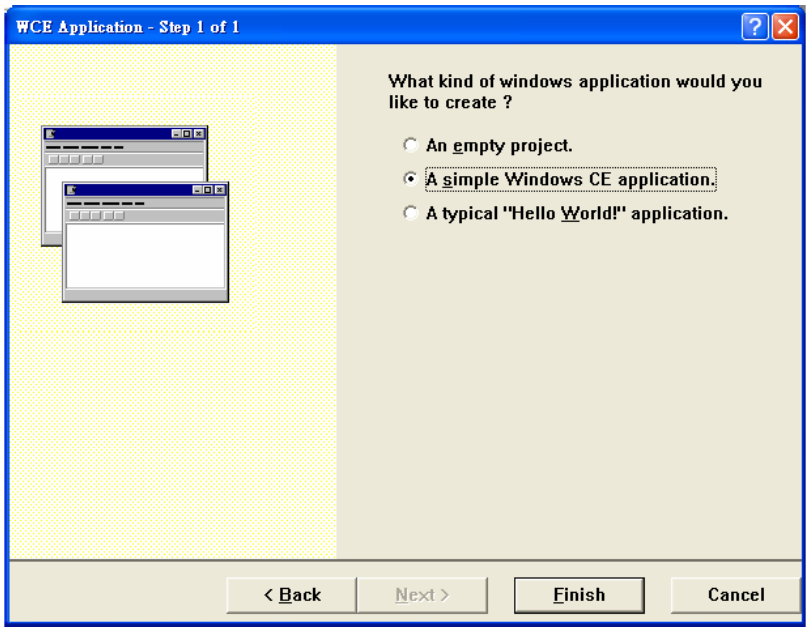
12. You may now start the debugging process.

Developing an eMbedded Visual C++ 4.0 Application

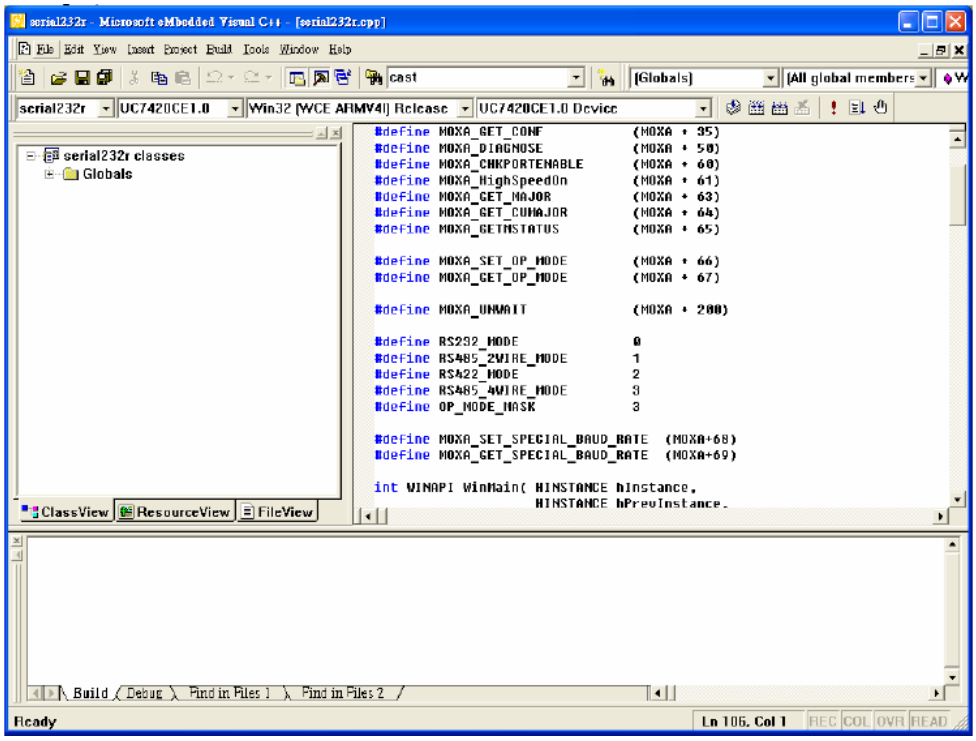
1. Before you perform on-line debugging, please make sure that no other ActiveSync device is connected to your workstation.
2. Open Microsoft® eMbedded Visual C++ 4.0.
3. From the **File** menu, choose **New.**
4. Choose the **Projects** tab and then select the type of application that you would like to build. Next, fill in the project name and then click **OK**.



5. Choose the type of application that you would like to create and then click **Finish.**



6. From the **Build** toolbar, choose the SDK (UC7400CE1.0 for this example), the type of run-time image (Release or Debug), and the target device (UC7400CE1.0 Device for this example).

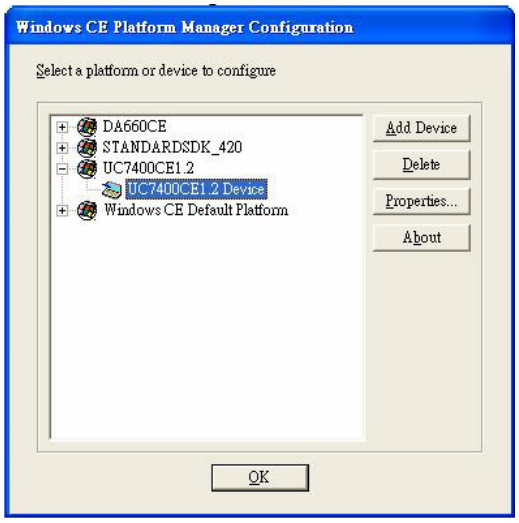


7. Write your code.
8. From the **Build** menu, choose **Rebuild All** to compile your application.
9. If the following window pops up, you may disregard it. The message is generated by ActiveSync while developing a mobile application. Your computer is not a mobile device.

10. When you complete your application, upload it to the embedded computer.
11. Log on the embedded computer and execute the program at the console prompt.

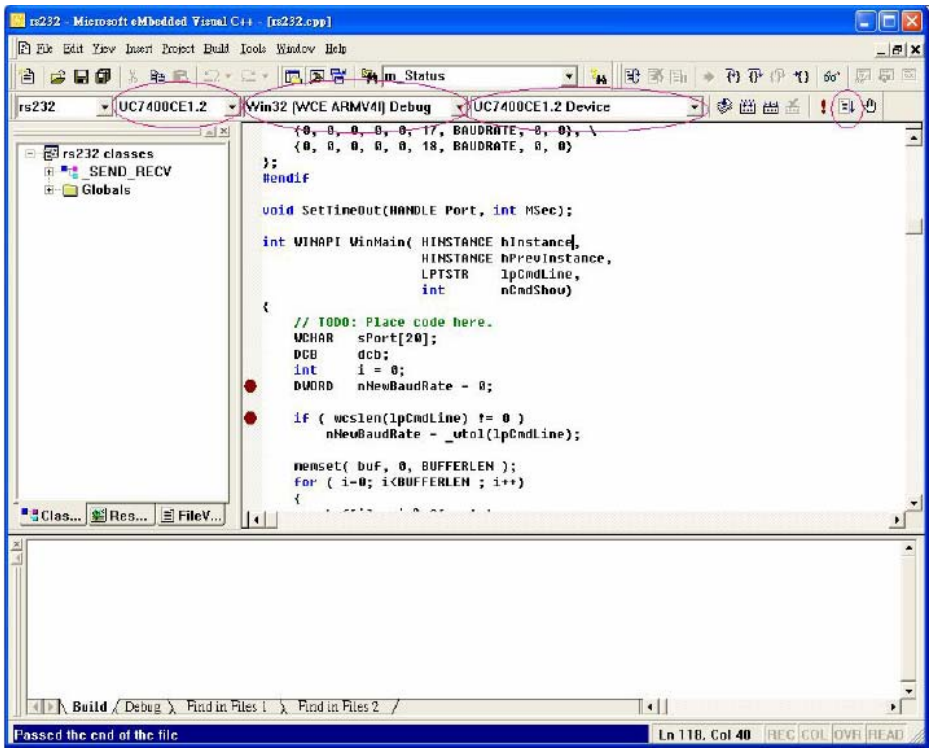
Debugging an eMbedded Visual C++ 4.0 Application

1. Open eMbedded Visual C++ 4.0.
2. Choose **Tools** on the main menu bar and then **Configure Platform Manager.**
3. Choose button **Properties.**

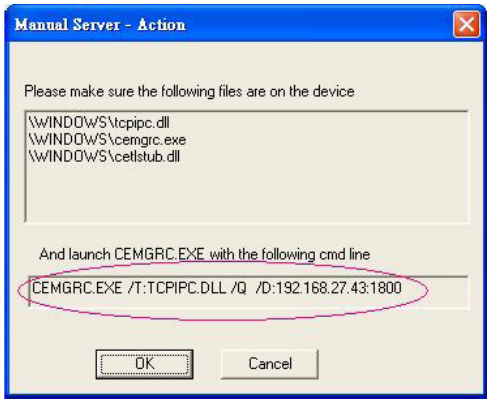


4. Select **TCP/IP Transport for Windows CE** from box **Transport** and select **Manual Server** from box **Startup Server.** Choose **OK.**

5. Back to eMbedded Visual C++ 4.0 and open or create a project.
6. Select the right SDK and the **Win32 (WCE ARMV4I) DEBUG** option.

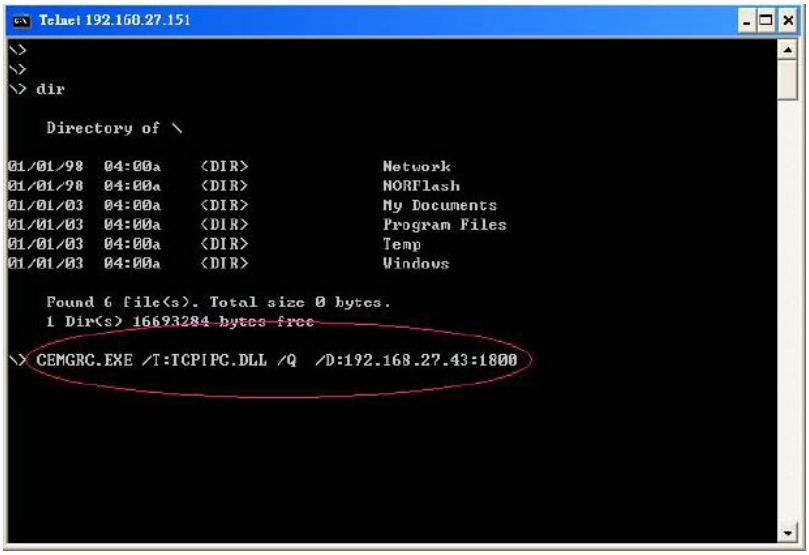


7. Set break points in your source code and press **F5** to start the on-line-debug.
8. On the popup **Manuel Server – Action** window, copy the string inside the text box. **NOTE: Do NOT click OK or Cancel at this time!**

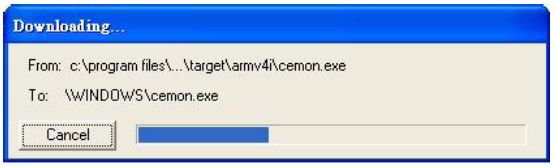


9. Log on the embedded computer using a Telnet client.

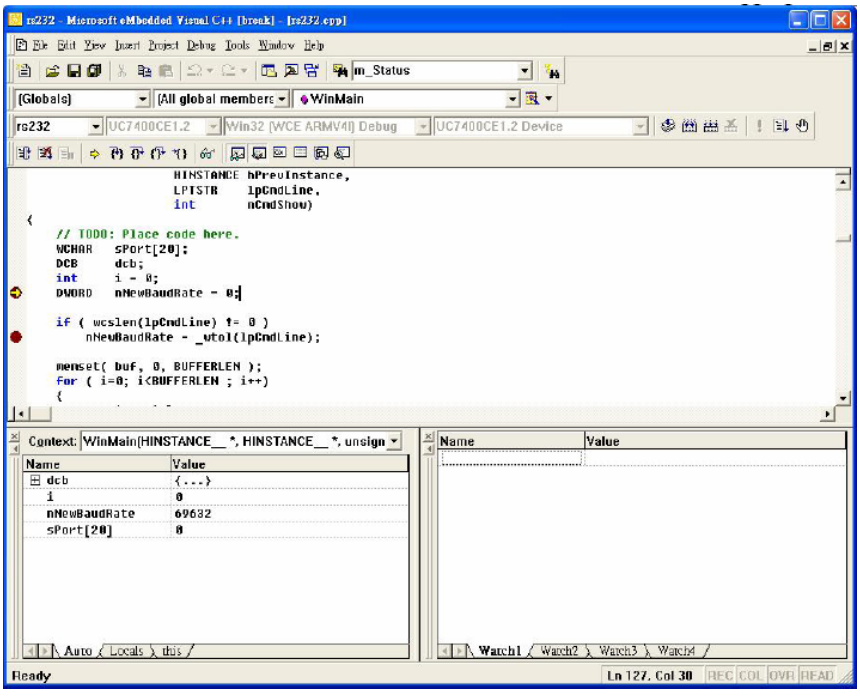
10. Paste the copied string to the Telnet window, and then press **Enter**.



11. Return to the **Manuel Server – Action** window and click **OK** to start a connection.
12. The following window will appear:



13. When a successful connection has been established, use **F10** for code debugging.



3

Programming Examples

In this chapter, we present

- **Before You Begin Programming**
  - Understanding the File System
  - Use the Flash Memory Cautiously
  - Storing Data in RAM vs. Flash Memory
  - Message Queue Programming
- **Visual C# Examples**
  - C# Example—Moxa UART (RS-232/422/485)
  - C# Example—Buzzer (UC-74XX-CE, DA-66X-CE, UC-712X-CE)
  - C# Example—Digital I/O (UC-7408-CE, IA-26X-CE, V468-CE)
  - C# Example—LCM (UC-7410-CE, UC-7420-CE)
  - C# Example—Function Keys (UC-7410-CE, UC-7420-CE)
  - C# Example—Real-time Clock (UC-74XX-CE, DA-66X-CE)
  - C# Example—TCP Server
  - C# Example—TCP Client
- **Visual C++ Examples**
  - C++ Example—Moxa UART (RS-232/422/485)
  - C++ Example—Buzzer (UC-74XX-CE, DA-66X-CE)
  - C++ Example—Digital I/O (UC-7408-CE, IA-26X-CE, V468-CE)
  - C++ Example—LCM (UC-7410-CE, UC-7420-CE)
  - C++ Example—Function Keys (UC-7410-CE, UC-7420-CE)
  - C++ Example—TCP Client and TCP Server
  - C++ Example—Message Queue

Before You Begin Programming

Understanding the File System

Before you begin programming your embedded computer, you should take time to understand the file system used by the embedded computer's operating system, and the characteristics of the storage media. This knowledge will help you make better use of the embedded computer's resources, and as a result, develop more efficient and effective applications.

RAM-based Storage

All of Moxa's embedded computers have a pre-partitioned RAM storage space mounted as a directory. For some models, this is just the root directory, and for other models, it's a directory named "RAMDisk". Files can be stored temporarily in this directory and its subdirectories, but they will be deleted when the computer is shut down or restarted. Persistent files and programs must be placed in the flash storage.

Flash Storage

NOR flash onboard memory storage is provided through the "NORFlash" directory. Data saved in this directory will be retained during a power failure or when the power is disconnected. CF (Compact Flash) and SD (Secure Digital) cards can also provide persistent NAND flash memory storage. These storage options provide more storage space than the onboard flash. We recommend that the onboard NOR flash be used for storing programs only. For log data generated by your programs, use external storage media such as CF or SD cards. These cards are much easier to replace if the card is damaged or the storage space is used up. Furthermore, there is much more storage space available on CF and SD cards than in the NOR flash memory.

Use the Flash Memory Cautiously

The onboard NOR flash memory has a life cycle of 100,000 write operations at the block level (each block can store 128 KB of data). Since the NOR flash does not support BBM (Bad Block Management), a FAT file system will not know whether or not the flash block has reached the end of its life cycle. As a result, the FAT file system will continue scanning a bad block again and again, eventually reaching an unstable state.

Since the FAT file system searches for free space sequentially when performing write operations, free storage space becomes more and more fragmented as files are deleted, making it difficult to search the space. Moreover, when a file is updated frequently, data is deleted and rewritten to the same flash blocks over and over again. Eventually, the FAT file system will be unable to read those blocks, causing the operating system to hang.

Although CF and SD cards also have life cycles, most cards use NAND flash memory in conjunction with hardware controllers to maximize the life cycle. The hardware controller acts as a bridge between the FAT file system and the memory space, and provides BBM (Bad Block Management) and wearing level management. The BBM feature allows the FAT file system to flag and skip any bad blocks. The wearing level feature works by writing data evenly over the entire area of the flash memory.

Storing Data in RAM vs. Flash Memory

Although data saved in RAM will be deleted when the system shuts down, RAM storage has the advantages of faster read/write access and no life cycle. For applications that require transmitting important data immediately and directly to a host, you can store the necessary log data in RAM. After the host receives the data, the data does not need to be retained and can be deleted.

Since embedded computers have resource limits, integrators must store data wisely. In general, you should only store data when you need to, and you should be sure to use the most appropriate storage medium.

Message Queue Programming

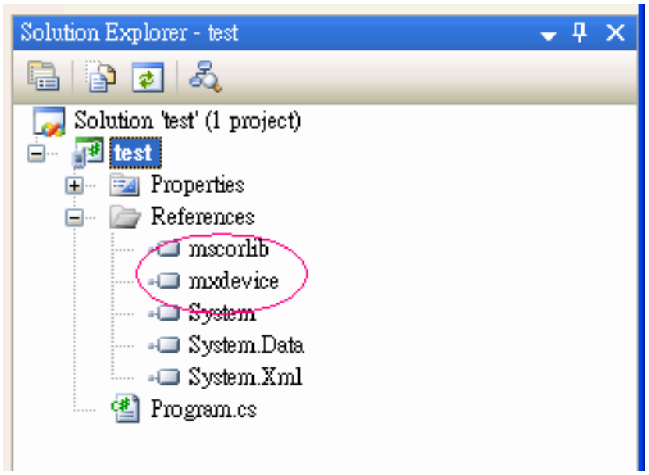
Under an ARM platform, file system reads/writes are a big load on the Windows CE operating system. In general cases of serial-to-Ethernet programming, we recommend using the Message Queue to receive serial data and avoid serial data receiving overrun problems.

There are special programming techniques associated with using Message Queue. We recommend keeping data block less then 1 KB, and wait 200 ms between every read/write. For details, please refer to example code.

Visual C# Examples

A device .Net CF 2.0 class library (*mxdevice.dll*) is provided to simplify application development with Visual Studio 2005 tools. This library covers the .Net CF Class Library for the buzzer, LCM, function keys, and digital I/O devices. To link the library with your Visual Studio 2005 project environment, perform the following steps from your Visual Studio 2005 tool:

1. Copy the library file **mxdevice.dll** to any folder on your local disk.
   This file can be found on the product CD in the folder **\sdk\dot Net Compact Framework Library**, or the file can be downloaded from the FTP site listed in the NOTE at the bottom of this page.
2. Open the Visual Studio 2005 IDE tool, and then add a new **C# Smart device console application**.
3. Enter the project name and location path.
4. In the **Solution Explorer View**, add **mxdevice.dll** to the reference section.
5. Click **OK.**



NOTE The programming examples for Moxa embedded computers are frequently updated. The latest examples can be downloaded from the following FTP site: ftp://esource.moxa.com/moxasys/WinCE_Examples/C#.

C# Example—Moxa UART (RS-232/422/485)

The following code is a C# sample program for the transmission of data to serial port COM3: This serial port can be configured to support RS-232, RS-422 or RS-485 operation mode. This program also receives messages from the serial port by using an event-based function.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Runtime.InteropServices;
using System.Collections.Specialized;
using System.IO.Ports;

namespace MxSerialPort
{
     class MxComPortConfig
     {
        [DllImport("coredll.dll", EntryPoint = "CreateFileW", SetLastError = true)]
        private static extern IntPtr CECreateFileW(string lpFileName, UInt32
        dwDesiredAccess, UInt32 dwShareMode, IntPtr lpSecurityAttributes, UInt32
        dwCreationDisposition, UInt32 dwFlagsAndAttributes, IntPtr hTemplateFile);

        [DllImport("coredll.dll", EntryPoint = "CloseHandle", SetLastError = true)]
        private static extern Int32 CECloseHandle(IntPtr hObject);

        [DllImport("coredll.dll", EntryPoint = "Sleep", SetLastError = true)]
        private static extern void CESleep(UInt32 dwMilliseconds);

        [DllImport("coredll.dll", EntryPoint = "DeviceIoControl",
        SetLastError = true)]
        public static extern bool DeviceIoControl(
             IntPtr hDevice,
             uint dwIoControlCode,
             ref byte lpInBuffer,
             int nInBufferSize,
             ref byte lpOutBuffer,
             int nOutBufferSize,
             ref int lpBytesReturned,
             IntPtr lpOverlapped
             );

        const uint MOXA_SET_OP_MODE = (0x400 + 66);
        const uint MOXA_GET_OP_MODE = (0x400 + 67);
        const uint RS232_MODE = 0;
        const uint RS485_2WIRE_MODE = 1;
        const uint RS422_MODE = 2;
        const uint RS485_4WIRE_MODE = 3;

        private readonly static IntPtr INVALID_HANDLE_VALUE = new IntPtr(-1);
        private const UInt32 OPEN_EXISTING = 3;
        private const UInt32 GENERIC_READ = 0x80000000;
        private const UInt32 GENERIC_WRITE = 0x40000000;

        String sPort;
        public MxComPortConfig(String sPort)
        {
             this.sPort = sPort;
        }
        private bool SetComPortInterface(uint mode)
        {
             IntPtr comPort = CECreateFileW(sPort, GENERIC_READ, 0, IntPtr.Zero,
```

```
            OPEN_EXISTING, 0, IntPtr.Zero);
            if (comPort == INVALID_HANDLE_VALUE)
                return false;

            int nBytesReturned = 0;
            byte bIn = (byte)mode;
            byte bOut = 0;
            DeviceIoControl(comPort, MOXA_SET_OP_MODE, ref bIn, 1, ref bOut, 0, ref
            nBytesReturned, IntPtr.Zero);
            CECloseHandle(comPort);
            return true;
        }

        public bool SetRS232()
        {
            return SetComPortInterface(RS232_MODE);
        }

        public bool SetRS422()
        {
            return SetComPortInterface(RS422_MODE);
        }

        public bool SetRS485TwoWire()
        {
            return SetComPortInterface(RS485_2WIRE_MODE);
        }

        public bool SetRS485FourWire()
        {
            return SetComPortInterface (RS485_4WIRE_MODE);
        }
    }

    class MxComPortEx
    {
        private SerialPort port;

        private MxComPortEx(string sPortName, int baudrate)
        {
            // create a SerialPort instance with basic settings port = new
            SerialPort(sPortName, baudrate, Parity.None, 8, StopBits.One);

            port.RtsEnable = true;        // raise RTS for flow control
            port.ReadTimeout = 3000;      // 3 second timeout

            // Attach a method to be called when there is data waiting in the port's
            buffer
            port.DataReceived += new SerialDataReceivedEventHandler(DataReceived);
            port.ErrorReceived += new SerialErrorReceivedEventHandler(ErrorEvent);
        }

        private void Open()
        {
            if (!port.IsOpen)
                port.Open();
        }

        private void Close()
        {
        if (port.IsOpen) port.Close();
        }

        private void SendData(String str)
        {
            Console.WriteLine("Sending Data: " + str);
            port.WriteLine(str);
        }
```

```
        private void ErrorEvent(object sender, SerialErrorReceivedEventArgs e)
        {
            Console.WriteLine("error");
        }

        private void DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            while (port.BytesToRead > 0)
            {
                Console.WriteLine("Receiving Data: " + port.ReadByte());
            }
        }

        private bool IsOpen()
        {
            return port. IsOpen;
        }

        static void Main(string[] args)
        {
            MxComPortEx comPort = new MxComPortEx("COM3:", 115200);

            comPort.Open();
            if (comPort. IsOpen())
            {
                int count = 0;
                while (count++ < 10)
                {
                    comPort.SendData("ABCDEFGH" + count);
                }
                comPort.Close();

            }
        }
    }
}
```

C# Example—Buzzer (UC-74XX-CE, DA-66X-CE, UC-712X-CE)

The embedded computer supports hardware buzzers that applications can use as an alarm for critical errors. You may set the frequency and the duration of the buzzer at the application level by using the APIs (within mxdevice.dll). The following example code triggers the buzzer for 50 milliseconds at frequency 2200 Hz.

Example Code for UC-74XX-CE and DA-66X-CE

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace test
{
    class Program
    {
        static void Main(string[] args)
        {
            moxa.uc7400.Buzzer.Beep(2200, 50);
        }
    }
}
```

Example Code for UC-712X-CE

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace test
{
   class Program
   {
        static void Main(string[] args)
        {
                  // start buzzer
                  moxa.uc712x.Buzzer.BeepOn();

                  // wait 20 milliseconds
                  System.Threading.Thread.Sleep(20);

                 // stop buzzer
                  moxa.uc712x.Buzzer.BeepOff();
        }
   }
}
```

C# Example—Digital I/O (UC-7408-CE, IA-26X-CE, V468-CE)

Digital input/output channels are featured in some models of embedded computer. These channels can be accessed at run-time for control or monitoring using the following example program.

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace test
{
   class Program
   {
        static void Main(string[] args)
     {
              int i=0;
              int port=0, value=0, value2=0;
              string sdout = "";
              string sin = System.Console.ReadLine();
              int n = int.Parse(sin);
              do
              {
                   switch (n)
                   {
                        case 1:
                          // display the digital input status.
                            for ( i=0; i<8; i++)
                            {
                                 value = moxa.uc7400.DigitalInput.Get(i);
                                 System.Console.WriteLine("Din{0}={1} ", i, value);
                            }
                            break;
                        case 2:
                          // display the digital output status.
                            for (i = 0; i < 8; i++)
                                 value = moxa.uc7400.DigitalOutput.Get(i);
                                 System.Console.WriteLine("DOut{0}={1}", i, value);
                            }
                            break;
                        case 3:
                          // set the digital output status
                            System.Console.Write("DOut port=");
```

```
                                sdout = System.Console.ReadLine();
                                port = int.Parse(sdout);

                                System.Console.Write("DOut value=");
                                sdout = System.Console.ReadLine();
                                value = int.Parse(sdout);

                                if (!moxa.uc7400.DigitalOutput.Set(port, value))
                                {
                                     System.Console.WriteLine("set dout failed!");
                                     break;
                                }

                                for (i = 0; i < 8; i++)
                                {
                                     value = moxa.uc7400.DigitalOutput.Get(i);
                                     System.Console.WriteLine("DOut{0}={1}", i, value);
                                }
                                break;
                           }
                      sin = System.Console.ReadLine();
                      n = int.Parse(sin);
                 } while (n != 0);
            }
       }
}
```

C# Example—LCM (UC-7410-CE, UC-7420-CE)

Certain models of embedded computer have an LCM, with text display managed by a pre-installed internal service. On these models, you can program a simple interface for your application. For UC-7400 models, the LCM is 8 × 16 characters; for DA-66x models, the LCM is 2 × 16 characters. Select the appropriate library and set the display location according to your model.

The following example transmits text to the LCM service on a UC-7400.

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace test
{
   class Program
   {
        static void Main(string[] args)
        {
          // Create the LCM obejct
             moxa.uc7400.LCM lcm = new moxa.uc7400.LCM();
          // display the LCM width and height
             System.Console.WriteLine("Width=" + lcm.width.ToString());
             System.Console.WriteLine("Height=" + lcm.height.ToString());
          // clear LCM screen
             lcm.cls();
          // write (append) message to the LCM
             lcm.write("abcdefg");
          // write message to the specific location. X=4, y=2
             lcm.write("1234567", 4, 2);
        }
   }
}
```

C# Example—Function Keys (UC-7410-CE, UC-7420-CE)

For embedded computers with programmable function keys, your application can drive them as shown in the following code examples. For example, your program can control a specified function key to trigger another application or switch your program between different operation modes. An internal service is pre-installed and mimics key presses for your application. Your application needs to register a key press with the service. Once the key press is detected, the service notifies your application via a software interrupt. Your application can then operate on your logic.

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace fkey
{
   class Program
   {
        private moxa.uc7400.KeyPad[] fkey;

        static void Main(string[] args)
        {
             int n = 0;

             // create the current class obejct
             Program keypadsvc = new Program();

             // wait 30 seconds
             while (n < 30)
             {
                  System.Threading.Thread.Sleep(1000);
                  n++;
                  System.Console.WriteLine(“.”);
             }

             // quit
             keypadsvc.stop();
             keypadsvc = null;

             System.Console.WriteLine(“exit...”);
        }

        public Program() {
             int       i=0;
             string    sKeyName;

             fkey = new moxa.uc7400.KeyPad[5];
             for ( i=1; i<=5; i++)
             {
               // make function key name
               sKeyName = “F” + i.ToString();
               // create function key object
               fkey[i-1] = new moxa.uc7400.KeyPad( sKeyName );
               // bind key click event to event function
               fkey[i-1].KeyClick += new
               moxa.uc7400.KeyClickEventHandler(keyOnclick);
               // start listening
               fkey[i-1].listen();
             }
        }

        public void stop()
        {
             int i=0;
             for ( i=0; i<5; i++)
```

```
            {
              // close function key object
              fkey[i].stop();
            }
        }

          ~Program()
        {
              System.Console.WriteLine("~Program()");
              int i=0;
              for ( i=0; i<5; i++)
              {
                // remove object from memory
                fkey[i]=null;
              }
              fkey=null;
          }

          public void keyOnclick (object sender, int e)
          {
              System.Console.WriteLine("key" + e.ToString() + " clicked");
          }
      }
}
```

C# Example—Real-time Clock (UC-74XX-CE, DA-66X-CE)

The real time clock (RTC) is a hardware implementation of time. It is different from the system time (or CPU time) that is generally used by applications to control timing issues. If your application needs the RTC value, you may copy and paste the following code into your application.

```
using System;
using System.Collections.Generic;
using System.Text;
using moxa;

namespace test
{
    class Program
    {
        static void Main(string[] args)
        {
            // get the RTC time
            moxa.mxdevice.SYSTEMTIME systime = moxa.uc7400.RTC.getRTC();
            System.Console.WriteLine("Year=" + systime.Year.ToString());
            System.Console.WriteLine("Month=" + systime.Month.ToString());
            System.Console.WriteLine("Day=" + systime.Day.ToString());
            System.Console.WriteLine("Hour=" + systime.Hour.ToString());
            System.Console.WriteLine("Minute=" + systime.Minute.ToString());
            System.Console.WriteLine("Second=" + systime.Second.ToString());
        }
    }
}
```

C# Example—TCP Server

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;

namespace socketsvr
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
                try
                {
                    if (args.Length == 0)
                    {
                        System.Console.WriteLine("e.g. tcpsvr [port]\n");
                        return;
                    }
                    // Set the TcpListener on port 13000.
                    Int32  port = int.Parse(args[0]);
                    System.Console.WriteLine("Listening port: " + args[0]);
                    TcpListener server = new TcpListener(port);

                    // Start listening for client requests.server.Start();
                    // Buffer for reading data
                    Byte[] bytes = new Byte[1024];
                    String data = null;
                    Byte[] rbytes = new Byte[1024];

                    Byte b1 = (Byte)'a';
                    for (int j = 0; j < 1024; j++)
                    {
                        bytes[j] = b1;
                        if (b1 == (Byte)'z')
                            b1 = (Byte)'a';
                        else
                            b1++;
                    }
                    // Perform a blocking call to accept requests.
                    // You could also user server.AcceptSocket() here.
                    TcpClient client = server.AcceptTcpClient();
                    data = null;
                    // Get a stream object for reading and writing
                    NetworkStream stream = client.GetStream();

                    Int32 i;
                    Int32 TotalCount = 10000000;
                    Int32 TotalBytes = 0;
                    while (true)
                    {
                        if (TotalCount == 0)
                            break;
                        // Loop to receive all the data sent by the client.
                        stream.Write(bytes, 0, bytes.Length);
                        while (stream. DataAvailable)
                        {
                            i = stream.Read(rbytes, 0, rbytes.Length);
                            TotalBytes += i;
                        }
                        TotalCount--;
                    }
                    // Shutdown and end connection
                    client.Close();
                    Console.WriteLine(String.Format("TotalBytes={0}, SendTimes={1}",
                    TotalBytes, TotalCount));
                }
                catch (SocketException e)
                {
                     Console.WriteLine("SocketException: {0}", e);
                }
            }
        }
    }
```

C# Example—TCP Client

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;

namespace socketcli
{
    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 2)
            {
                System.Console.WriteLine("socketcli [ip] [port]");
                return;
            }
            IPHostEntry myIpHostEntry = Dns.GetHostEntry( args[0] );
            Int32 port = Int32.Parse(args[1]);
            IPEndPoint myIpEndPoint = new
            IPEndPoint(myIpHostEntry.AddressList[0], port);

            Socket mySocket = new
            Socket(myIpEndPoint.Address.AddressFamily,
                                        SocketType.Stream,
                                        ProtocolType.Tcp);
            UInt32 TotalBytes = 0;
            try
            {
                 mySocket.Connect(myIpEndPoint);

                 // Create the NetworkStream for communicating with the remote
                host.
                NetworkStream myNetworkStream;

                if (mySocket.Connected)
                {
                     myNetworkStream = new NetworkStream(mySocket, true);
                }
                else
                {
                     myNetworkStream = new NetworkStream(mySocket);
                }

                byte[] myReadBuffer = new byte[1024];
                int numberOfBytesRead = 0;
                Int32 dwCount=0;

                while ( myNetworkStream.CanRead)
                {
                    // Incoming message may be larger than the buffer size.
                    do
                    {
                        numberOfBytesRead =
                        myNetworkStream.Read(myReadBuffer, 0,
                        myReadBuffer.Length);
                        TotalBytes += (UInt32)numberOfBytesRead;
                    }
                    while (myNetworkStream.DataAvailable);
                    dwCount++;
                    if (dwCount % 1000 == 0)
                          Console.WriteLine("Received Total=" +
                          TotalBytes.ToString());

                    if (myNetworkStream.CanWrite)
                    {
```

```
                        myNetworkStream.Write(myReadBuffer, 0,
                        myReadBuffer.Length);
                    }
                }
                // Close the NetworkStream
                myNetworkStream.Close();

            }
            catch (Exception exception)
            {
                Console.WriteLine("Exception Thrown: " + exception.ToString());
            }
            Console.Write(Console.Read());
            Console.Read();
            }
    }
}
```

Visual C++ Examples

A library (mxdev.lib) is provided to simplify application development. This library covers APIs for the buzzer, LCM, function key, and digital I/O devices.

To link it with your VS2005 developing environment, perform the following steps from your VS2005 tool:

1. From the VS2005 solution explorer, right-click the project and choose **Property**.
2. From the Left window, choose **Configuration Properties → Linker → Input.**
3. Append *mxdev.lib* to the text field **Additional Dependencies.**

To link it with your eVC4.0 developing environment, perform the following steps from your eVC4.0 tool:

1. From the main tool bar, choose **Project → Settings.**
2. From **Project Settings,** select the **Link** tab.
3. Append *mxdev.lib* to the text field **Object/Library Modules:**
4. Click **OK**

Before you compile your application, please make sure that the header file and the library file in the SDK directories are as shown below. If any of them is not in the specified directory, find it on the package CD and copy it to the specified directory.

"C:\Program Files\Windows CE Tools\wce500\<model version>\
UC7420CE1.0\Include\Armv4i\moxa\devices.h"

"C:\Program Files\Windows CE Tools\wce500\<model version>\
UC7420CE1.0\lib\Armv4i\mxdev.lib"

NOTE The programming examples for Moxa embedded computers are frequently updated. The latest examples can be downloaded at ftp://esource.moxa.com/moxasys/WinCE_Examples/C++.

C++ Example—Moxa UART (RS-232/422/485)

The following C/C++ code shows a sample application that transmits text data from port COM3 to port COM4 in the RS-232 operation mode. The application opens these two ports, generates a thread to receive data from port COM4, and then executes as a main thread to transmit data to port COM3.

```
#include "stdafx.h"
#include <stdio.h>
#include <moxa/devices.h>

#define MAX_DATA_LEN 128
//=========================================================================
static HANDLE createComHandle(WCHAR *comPort, unsigned int baudrate)
{
  HANDLE hCom;
  DCB dcb;
  COMMTIMEOUTS    to;

  hCom = CreateFile(comPort, GENERIC_READ | GENERIC_WRITE,
              0, // exclusive access, until the handle is closed
              NULL, // no security
              OPEN_EXISTING,
              0, // no overlapped I/O
              NULL); // null template
              if (hCom== INVALID_HANDLE_VALUE )
              return NULL;

  // set serial setting
  GetCommState(hCom, &dcb);
  dcb.BaudRate = baudrate;
  dcb.ByteSize = 8;
  dcb.StopBits = ONESTOPBIT;
  dcb.Parity = NOPARITY;
  dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
  dcb.fOutxCtsFlow = TRUE;
  SetCommState(hCom, &dcb);

  // set timeout parameter
  to.ReadIntervalTimeout = 0;
  to.ReadTotalTimeoutMultiplier = 0;
  to.ReadTotalTimeoutConstant = 0;
  to.WriteTotalTimeoutMultiplier = 0;
  to.WriteTotalTimeoutConstant = 0;
  if (!SetCommTimeouts(hCom,&to))
  {
              printf("SetCommTimeouts error!\n");
              CloseHandle(hCom);
              return NULL;
  }
  return hCom;
}
//=======================================================
static DWORD comReadThread(LPVOID param)
{
  HANDLE hCom = (HANDLE)param ;
  DWORD rtn;
  unsigned char buffer[MAX_DATA_LEN+1];
  while(1)
  {
              if (ReadFile(hCom, buffer, MAX_DATA_LEN, &rtn, NULL)==0)
              {
                        printf("read data fail\n");
                        return 0;
              }
              buffer[rtn] = '\0';
              printf("Data = %s\n", buffer);
```

```
  }
  CloseHandle(hCom);
}
int
comPair(WCHAR *wComPort, WCHAR *rComPort, unsigned int baudrate) {
  HANDLE wCom, rCom;
  WCHAR sPort[64];
  DWORD rtn, i, loop=0;
  unsigned char buffer[MAX_DATA_LEN];
  HANDLE waitH = CreateEvent(NULL, FALSE, FALSE, NULL);

  if (waitH==NULL)
            return 99;

  /* create a handle to port "COM3" for transmitting data */
            wsprintf(sPort, L"$device\\%s\0", wComPort);
  wCom = createComHandle(sPort, baudrate);
  if (wCom==NULL)
  {
            printf("Fail to create write port\n" );
            return 1;
  }

  /* create a handle to port "COM3" for receiving data */
  wsprintf(sPort, L"$device\\%s\0", rComPort);
  rCom = createComHandle(sPort, baudrate);
  if (rCom == NULL)
  {
            printf("Fail to create read port\n");
            CloseHandle(wCom);
            return 2;
  }

  /* for a thread to handle receiving */
  if (CreateThread( NULL, 0, comReadThread, (LPDWORD) rCom, NULL, 0 )==NULL)
  {
            printf("Fail to create a receiving thread\n");
            CloseHandle(wCom);
            CloseHandle(rCom);
            return 3;
  }

  for (i=0; i< MAX_DATA_LEN;i++)
            buffer[i] = (unsigned char) ('a'+i%26);

  PurgeComm(wCom, PURGE_TXCLEAR | PURGE_TXABORT);
  while(loop++ < 100)
  {
        if (WriteFile(wCom, buffer, MAX_DATA_LEN, &rtn, NULL)==0)
            {
                        printf("Fail to write\n");
                        break;
            }
            WaitForSingleObject(waitH, 100);
  }
  CloseHandle(wCom);
  CloseHandle(rCom);
  CloseHandle(waitH);

  return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPTSTR lpCmdLine, int nCmdShow)
{
  comPair(L"COM3", L"COM4", 38400);
  return 0;
}
```

Windows® API function **CreateFile(...)** opens a named file corresponding to a serial port. For serial ports COM10 or after, the file name must be prefixed by "$device\\". For ports COM1 to COM9, the file name may be prefixed by "$device\\" but it is not required. API function **GetCommState(..)** is the standard function to obtain a serial port's current parameters, such as baudrate, data bits (bytesize),parity and stop bits. API function **SetCommState(...)** is used to set those parameters. Other parameters, including flow control, DTR signal, and RTS signal, can be adjusted to fine-tune the serial port. Be sure to set the flow control correctly or data will be lost.

The embedded computer supports four serial communication interfaces: RS-232 (default), RS-422, 2-wire RS-485, and 4-wire RS-485. To change the operation mode, your program should include the following macro definitions. The DeviceIoControl function should be inserted on the open handle before the program performs read/write operations.

```
/* --------- extracted from <moxa/devices> -------------------
#define MOXA_SET_OP_MODE    (0x400 + 66)
#define MOXA_GET_OP_MODE    (0x400 + 67)
#define RS232_MODE              0
#define RS485_2WIRE_MODE    1
#define RS422_MODE              2
#define RS485_4WIRE_MODE    3
*/

/* for example, set the mode to 4-wire RS485 */
BYTE mode = RS485_4WIRE_MODE;
DeviceIoControl(hCom,MOXA_SET_OP_MODE, &mode, sizeof(mode),
NULL,0,NULL,NULL);
/* for example, get the current mode */
BYTE mode, size;
DeviceIoControl(hCom,MOXA_GET_OP_MODE, NULL, 0, (LPVOID)&mode,
sizeof(mode),&size,NULL);
```

C++ Example—Buzzer (UC-74XX-CE, DA-66X-CE)

The embedded computer supports hardware buzzers that applications can use as an alarm for critical errors. You may set the frequency and the duration of the buzzer at the application level by using the APIs (within mxdev.lib). The following example code triggers the buzzer for 2000 milliseconds at frequency 500Hzs.

```
/* execute a beeper at a specified frequency for lasting a duration in
miniseconds */
BOOL mxbeep(unsigned int nHz, unsigned int nMiniSec);

#include <moxa/devices.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPTSTR       lpCmdLine, int      nCmdShow)
{
mxbeep(500,2000);
return 0;
}
```

C++ Example—Digital I/O (UC-7408-CE, IA-26X-CE, V468-CE)

The embedded computer with the support of programmable digital input/output channels provides you with I/O control on other devices. These channels can be accessed at run-time via the following example program.

```
/* this function initialize a connection to the devices. It must be called
before you call other functions
        On success, it returns a handle. Otherwise, it return NULL
*/
HANDLE mxdio_init(void);

/* this function requests the system to detect a low-to-high or high-to-low
event on a specified input port every tick_slot milliseconds. On success, it
```

```
returns 0.
<hndl> handle created by mxdio_init
<port> port number 0~7
<handler> callback function with prototype (unsigned int, unsigned int,
unsigned int)
<low_high> macro DIN_EVENT_LOW_TO_HIGH or DIN_EVENT_HIGH _TO _LOW
<tick_slot> 10 milliseconds minimal
*/
int mxdio_set_input_event( HANDLE hndl,
unsigned int port,
mxdio_input_cb handler,
unsigned int low_high,
unsigned int tick_slot);

/* this function starts a dispatcher to handle events.
<hndl> handle created by mxdio_init
Return non-zero to abort.
*/
int mxdio_dispatch(HANDLE hndl);

/* this function set a digit level for a specified output port
<hndl> handle created by mxdio_init
<port> port number 0~7
<data> 0 or 1
*/
int mxdio_set_dout(HANDLE hndl, unsigned int port, unsigned int data);

/* this function gets the current digit level of a specified port
<hndl> handle created by mxdio_init
<port> port number 0~7
Return DIN_EVENT_LOW_TO_HIGH or DIN_EVENT_HIGH_TO_LOW (0 or 1)
*/
int mxdio_get_dout(HANDLE hndl, unsigned int port);
```

To use the library **mxdev.lib**, include file **moxa/devices.h** in your program. The following example shows a simple mechanism to use the library.

```
#include <moxa/devices.h>

DIOHANDLER(port0_process)
{
printf("port_0_process = %d\t%d\t%ld\n", port, type,syst_tick);
}

DIOHANDLER(port1_process)
{
printf("port_1_process = %d\t%d\t%ld\n", port, type,syst_tick);
}

int WINAPI WinMain( HINSTANCE hInstance,HINSTANCE hPrevInstance,
LPTSTR lpCmdLine,int nCmdShow)
{
HANDLE h = mxdio_init();
if(h==NULL)
{
            printf("fail to init\n");
            return 1;
}

mxdio_set_input_event(h, 0, port0_process, DIN_EVENT_LOW_TO_HIGH, 30);
mxdio_set_input_event(h, 1, port1_process, DIN_EVENT_HIGH_TO_LOW, 20);
mxdio_dispatch(h);
return 0;
}
```

All the callback functions have the same format (*unsigned int port*, *unsigned int type*, *unsigned sys_tick*). Name each of the functions and enter the logic of your application inside it.

In the example shown above, the dispatcher checks digit level changes every 20 milliseconds and every 30 milliseconds from the moment that the dispatcher function *mxdio_dispatch()* is called. If there is any change, the associated callback function is initiated.

C++ Example—LCM (UC-7410-CE, UC-7420-CE)

Certain models of embedded computer have an LCM, with text display managed by a pre-installed internal service. On these models, you can program a simple interface for your application. For UC-7400 models, the LCM is 8 × 16 characters; for DA-66x models, the LCM is 2 × 16 characters. Select the appropriate library and set the display location according to your model.

The following example transmits text to the LCM service on a UC-7400.

```
#include <windows.h>
#include <moxa/devices.h>

int WINAPI WinMain( HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPTSTR  lpCmdLine,
int     nCmdShow)
{
HANDLE hndl;
char *buf="Hi, hello World!!\nabcdefg\n\n1234567890";
unsigned int w, h, x, y;

hndl = mxlcm_open(&w, &h);
if (!hndl)
   return 1;

x = 2; y=1;
mxlcm_clear(hndl);
mxlcm_write(hndl, buf, strlen(buf), &x, &y);
mxlcm_close(hndl);
return 0;
}
```

Upon detecting the character '\n' or '\r' in the received text, the internal service forces a down shift of a line on the display screen. In addition, the service scrolls up a lengthy portion of text by dropping its leading part in order to fit the area of the LCM display.

```
/* open a handle to the LCM devie
   <lcm_w> the function outputs the width of the LCM display to this pointer
   <lcm_h> the function outputs the height of the LCM display to this pointer
   Return nonzero value in success.
*/
HANDLE mxlcm_open(unsigned int *lcm_w, unsigned int *lcm_h);

/* write text to the LCM device at a specify coordinate
   <hndl> the open handle
   <buffer> text
   <len> the length of the text
   <lcm_x> and <lcm,_y> pointers to the coordinate of the starting position
   of the text,
   the function would output the ending coordinate of the text to these
   pointers.
   This function returns the number of written bytes
*/
int mxlcm_write(HANDLE hndl, char *buffer, unsigned int len, unsigned int
*lcm_x, unsigned int *lcm_y);

/* close the handle */
void mxlcm_close(HANDLE hndl);

/* clear the LCM screen */
void mxlcm_clear(HANDLE hndl);
```

C++ Example—Function Keys (UC-7410-CE, UC-7420-CE)

For embedded computers with programmable function keys, your application can drive them as shown in the following code examples. For example, your program can control a specified function key to trigger another application or switch your program between different operation modes.

An internal service is pre-installed and mimics key presses for your application. Your application needs to register a key press with the service. Once the key press is detected, the service notifies your application via a software interrupt. Your application can then operate on your logic.

In the application level, library mxdev.lib offers the following APIs.

```
#define FKEYHNDL(f) int f(void* param)

/* initialization */
int     mxfkey_init(void);

/* stop */
void    mxfkey_stop(void);

/* register an operation to a keypad
   <key> a pointer to a string "F1"~"F5"
   <cb> a callback function to be trigger when the associated keypad is pressed
   <param> parameters that would be carried to the callback function
   Return zero in success.
*/
int     mxkey_register(char *key, FKEYHNDL ((*cb)), void *param);
```

All you need to do is register your callback function to an associated function key on the keypad, using the format (unsigned int key, void* param). The argument param is used to transfer your private data to the callback function.

```
#include <windows.h>
#include <moxa/devices.h>

FKEYHNDL(keyF2)
{
   printf("keyF2 is pressed\n");
   return 0;
}

FKEYHNDL(keyF3)
{
   printf("keyF3 is pressed\n");
   return 0;
}

int WINAPI WinMain( HINSTANCE hInstance,
   HINSTANCE hPrevInstance,
   LPTSTR    lpCmdLine,
   int       nCmdShow)
{
   HANDLE hEvent;
   DWORD rtn;

   if (mxfkey_init())
return 1;
   rtn = mxfkey_register("F2", keyF2, 0);
   if (rtn)
return 1;
   rtn = mxfkey_register("F3", keyF3, 0);
   if (rtn)
   {
mxfkey_stop();
return 1;
   }
```

```
   hEvent = CreateEvent( NULL, FALSE, FALSE, NULL );
   WaitForSingleObject( hEvent,20000 );
   mxfkey_stop();
   return 0;
}
```

C++ Example—TCP Client and TCP Server

The following is an example of TCP client/server programming. To build a server program, compile this sample code in the eVC4.0++ tool with library ws2.lib to build a client program, compile same exact code in a Windows development tool, such as Visual Studio 6.0 with library wsock32. lib.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#include <windows.h>
#include <winsock.h>
#include <stdarg.h>

#ifdef _WIN32_WCE
#define RNADOM_SEED GetTickCount()
#define ipc_create_thread(t,a) CreateThread(NULL, 0, t, a, 0, NULL)
#else

#include <sys/types.h>
#include <sys/stat.h>
#include <process.h>
#include <errno.h>
#define RNADOM_SEED time(0)
#define ipc_create_thread(t,a) _beginthread(t, 0, a)
#endif

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif

#define MAX_BUF_LEN 1024
#define MAX_SOCKETS 64

typedef struct _HINFO
{
   char hostname[64];
   int       port;
   int  fd;
} HINFO;

typedef struct _SOCKCONN
{
   int  port;
   int  fd;
} SOCKCONN;

/*
 * Global Variables
 */
typedef struct _global_t
{
   fd_set    read_fds;
   int       max_fd;
   SOCKCONN  clients[MAX_SOCKETS];
   int       client_num;
   SOCKCONN  servers[MAX_SOCKETS];
   int       server_num;
} global_t;
```

```
global_t global;

#ifndef _WIN32_WCE
/* --------- client program -------------------- */
static unsigned int send_bytes=0;

int
test_client_thread(LPVOID param)
{
static char
*alphaNumber="abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   int fd = (int) param;
   char buf[MAX_BUF_LEN];
   int i,len;

   srand(RNADOM_SEED);
   while (send_bytes<0xffffffff)
   {
        len = rand()%1024;
        if (len ==0) continue;
        for (i=0; i < len; i++)  buf[i] = alphaNumber[rand()%62];
        buf[i] = 0;
        send(fd, buf, len, 0);
        send_bytes += len;
printf("\r%lu", send_bytes);
        Sleep (200);
   }
printf("test_client_thread\n");
   return 0;
}

void
handle_packet(SOCKCONN *s, char *buf, int len)
{
   static unsigned int num_bytes=0;
   int fd = s->fd;

   buf[len]=0;
   if (send_bytes)
   {
   num_bytes+=(unsigned int) len;
   printf("\r%lu/%lu", num_bytes, send_bytes);
   }
   else
   printf("[%3d] %s\n", len, buf);
}

/*
   argv: <host> <port>...
*/
int
clients_init(int argc, char **argv, HINFO *hosts, int max)
{
   int i,n;
   if (argc < 2)
   {
        printf("Usage: progname <host> <TCPPort>\n");
        exit(1);
   }
   for (i = 0, n = 0; i < argc && n < max; i+=2,n++)
   {
        strcpy(hosts[n].hostname, argv[i]);
     hosts[n].port = atoi (argv[i+1]);
   }
   return n;
   }

/*
```

```
   argv: <host> <port>...
*/
void
clients_post_init(int argc, char **argv, HINFO *hosts, int max)
{
   int i;
   for (i=0; i < max; i++)
   {
        ipc_create_thread(test_client_thread, (void*)hosts[i].fd);
   }
}

int
servers_init(int argc, char **argv, HINFO *hosts, int max)
{
   return 0;
}

void
servers_post_init(int argc, char **argv, HINFO *hosts, int max)
{
}

#else /* _WIN32_WCE */
/* --------- server program ------------------- */

void
handle_packet(SOCKCONN *s, char *buf, int len)
{
   static unsigned int num_bytes=0;

   num_bytes+=(unsigned int) len;
   printf("\r%lu", num_bytes);
}

/*
   argv: <port>...
*/
int
servers_init(int argc, char **argv, HINFO *hosts, int max)
{
   int i;

   if (argc < 1)
{
   printf("Usage: progname <TCPPort>\n");
   exit(1);
}
   for (i = 0; i < argc && i < max; i++)
        hosts[i].port = atoi (argv[i]);
   return i;
}

void
servers_post_init(int argc, char **argv, HINFO *hosts, int max)
{
}

int
clients_init(int argc, char **argv, HINFO *hosts, int max)
{
   return 0;
}

void
clients_post_init(int argc, char **argv, HINFO *hosts, int max)
{
}
```

```
#endif

unsigned int
lookup_ip (const char *host)
{
   struct hostent *he;
   unsigned int ip;

   ip = inet_addr (host);
   if (ip == INADDR_NONE)
{
   he = gethostbyname (host);
   if (!he)
   {
        printf ("lookup_ip: can't find ip for host %s\n", host);
        return 0;
   }
   memcpy (&ip, he->h_addr_list[0], he->h_length);
   }
   return ip;
}

/* set socket to be nonblocking */
int
tcp_set_nonblocking (int f)
{
   long val = 1;
   if (ioctlsocket (f, FIONBIO, &val) != 0)
   {
   printf ("tcp_set_nonblocking: fcntl\n");
   return -1;
   }
   else
   return 0;
}

/* turn on TCP/IP keepalive messages */
int
tcp_set_keepalive (int f, int on)
{
   if (setsockopt(f, SOL_SOCKET, SO_KEEPALIVE, (void*) & on, sizeof (on))
   == -1)
   {
        printf ("set_keepalive: setsockopt\n");
        return -1;
   }
   return 0;
   }

/* a nonblocking and keepalive socket */
static int
new_socket (void)
{
   int       f;

   f = socket (AF_INET, SOCK_STREAM, 0);
   if (f < 0)
        return -1;
   if(tcp_set_nonblocking (f))
   {
        closesocket (f);
        return -1;
   }
   if(tcp_set_keepalive (f,1))
   {
        closesocket (f);
        return -1;
   }
```

```
   return f;
}

/* bind a socket on a specific interface */
static int
bind_interface (int fd, unsigned int ip, int port)
{
   struct sockaddr_in sin;

   memset (&sin, 0, sizeof (sin));
   sin.sin_family = AF_INET;
   sin.sin_addr.s_addr = ip? ip:htonl(INADDR_ANY);
   sin.sin_port = htons ((unsigned short) port);
   if (bind (fd, (struct sockaddr *) &sin, sizeof (sin)) < 0)
   {
        printf ("bind_interface: bind\n");
        return -1;
   }
   return 0;
}

/* a client connection */
int
make_client_connection (const char *host, unsigned short port, unsigned int
*ip)
{
        struct sockaddr_in sin;
        int  f;

        memset (&sin, 0, sizeof (sin));
        sin.sin_port = htons (port);
        sin.sin_family = AF_INET;

        if (!host && !ip)
             return -1;
        if (host)
             *ip = lookup_ip (host);
        if (*ip == 0)
             return -1;
        sin.sin_addr.s_addr = *ip;

        if ((f = new_socket ()) == -1)
             return -1;

        printf("connecting to %s at port %hu\n",inet_ntoa (sin.sin_addr), ntohs
        (sin.sin_port));
        if (connect (f, (struct sockaddr *) &sin, sizeof (sin)) < 0)
        {
             if (h_errno != WSAEINPROGRESS && h_errno != WSAEWOULDBLOCK)
             {
                  closesocket (f);
                  return -1;
             }
             printf ("nonblocking yet connected to %s\n", host);
        }
        else
             printf ("connection ok to %s\n", host);
        return f;
}

int
tcp_startup_server(unsigned int iface, int port)
{
   int sockfd;
        struct in_addr addr;

   if ((sockfd = new_socket ()) < 0)
        return 0;
```

```
   if (bind_interface (sockfd, iface, port) == -1)
        return 0;
   if (listen (sockfd, 5) < 0)
        return 0;
   addr.s_addr = iface;
printf("startup a server %s at port %d\n",inet_ntoa (addr),port);
   return sockfd;
}

void
enable_read (int fd)
{
   FD_SET ((unsigned int) fd, &global. read_fds);
   if (global.max_fd < fd)  global.max_fd = fd;
}

void
disable_read (int fd)
{
   FD_CLR ((unsigned int) fd, &global. read_fds);
}

void
accept_connection(int fd)
{
   unsigned int sinsize;
   struct sockaddr_in sin;
   sinsize = sizeof (sin);
   fd = accept (fd, (struct sockaddr *) &sin, &sinsize);
   if(fd>0)
   {
        enable_read(fd);
        global.clients[global.client_num++].fd = fd;
printf("accept a client connection fd %d\n", fd);
   }
}

void
remove_connection(SOCKCONN *s)
{
printf("remove_connection a client connection fd %d\n", s->fd);
   disable_read(s->fd);
   closesocket(s->fd);
   *s = global.clients[--global.client_num];
}

#if defined(_WIN32_WCE)
int WINAPI
WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow )
#else
int
main (int argc, char **argv)
#endif
{
   struct timeval to;
   unsigned int ip;
   int  i, fd, len, num_ports;
   char buf[MAX_BUF_LEN];
   HINFO     hosts[MAX_SOCKETS];
   fd_set    read_fds;
   WSADATA wsa;

#if defined(_WIN32_WCE)
   int  argc;
   char cmdline[256], *argv[32], *p;

   WideCharToMultiByte(CP_ACP, 0, (LPCTSTR)lpCmdLine, 255, cmdline, 256, NULL, NULL);
```

```
    p = cmdline;
    argc=1;
    while (argc < 32)
    {
         while(isspace(*p)) p++;
         if (*p==0)
              break;
         argv[argc++] = p;
         p = strchr(p, ' ');
         if (!p)
              break;
         *p = 0;
         p++;
    }
#endif

    WSAStartup (MAKEWORD (2, 2), &wsa);
    memset(&global, 0, sizeof(global_t));

    memset(hosts, 0, MAX_SOCKETS*sizeof(HINFO));
    /* obtain listen ports from user-defined function */
    num_ports = servers_init(argc-1, argv+1, hosts, MAX_SOCKETS);
    for (i = 0; i < num_ports; i++)
    {
         fd = tcp_startup_server(0, hosts[i].port);
         if (fd <= 0)
              continue;
         enable_read(fd);
         global.servers[global.server_num++].fd = hosts[i].fd = fd;
    }
    servers_post_init(argc-1, argv+1, hosts, num_ports);

memset(hosts, 0, MAX_SOCKETS*sizeof(HINFO));
/* obtain hosts and their respective ports from user-defined function */
num_ports = clients_init(argc-1, argv+1, hosts, MAX_SOCKETS);
for (i = 0; i < num_ports; i++)
{
    fd = make_client_connection(hosts[i].hostname, (unsigned short)
    hosts[i].port, &ip);
    if (fd <= 0)
         continue;
    enable_read(fd);
    global.clients[global.client_num++].fd = hosts[i].fd = fd;
}
clients_post_init(argc-1, argv+1, hosts, num_ports);

/* main event loop */
while (1)
{
    read_fds = global.read_fds;

    to.tv_sec = 2;
    to.tv_usec = 0;
    if (select (global.max_fd + 1, &read_fds, 0, 0, &to) == -1)
         break;

    /* do client i/o sequentially */
    for (i = 0; i < global.client_num; i++)
    {
         fd = global.clients[i].fd;
         if (!FD_ISSET(fd,&read_fds))
              continue;

         if ((len = recv(fd, buf, MAX_BUF_LEN, 0)) <= 0)
              remove_connection(&global.clients[i]);
         else
              handle_packet(&global.clients[i], buf, len);
    }
```

```
   /* handle connection requests */
   for (i = 0; i < global.server_num; i++)
   {
        fd = global.servers[i].fd;
        if (FD_ISSET(fd,&read_fds))
             accept_connection(fd);
   }
}
/* close all open file descriptors properly */
   for (i = 0; i < global.client_num; i++) closesocket
   (global.clients[i].fd);
   for (i = 0; i < global.server_num; i++) closesocket
   (global.servers[i].fd);

#ifdef WIN32
   WSACleanup ();
#endif
   return 0;
```

C++ Example—Message Queue

```
// MessageQueue.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <windows.h>
#include <commctrl.h>

#define QUEUE_NAME L"MQ_DEMO"

struct _DATA_t {
   unsigned int id;
   BYTE *pData;
   DWORD dwLen;
};

BOOL bQuit = FALSE;
CRITICAL_SECTION cs;

int getRand(int nID);

#define CLIENTS 5

DWORD thread_main(LPVOID PPARAM)
{
   MSGQUEUEOPTIONS msq;
   _DATA_t m_data;
   DWORD dwRead, dwflags;
   DWORD dwDataRecvCount[CLIENTS];
   DWORD dwDataRecvBytes[CLIENTS];

   for ( int i=0; i<CLIENTS; i++) {
        dwDataRecvCount[i] = 0;
        dwDataRecvBytes[i] = 0;
   }

   msq.dwSize = sizeof(MSGQUEUEOPTIONS);
   msq.bReadAccess = TRUE;               // Read permission
   msq.dwMaxMessages = 0;               // Set the max messages number of message queue
to be INFINITE
   msq.dwFlags =  MSGQUEUE_NOPRECOMMIT | MSGQUEUE_ALLOW_BROKEN;
   msq.cbMaxMessage = sizeof(_DATA_t);     // assign to message data structure size

   // create a read only message queue
   HANDLE hMsq = CreateMsgQueue(QUEUE_NAME, &msq);
   printf("Press [Enter] to leave program!! ...\n");
```

```
   while (TRUE)
   {
        // waiting for data goes into the Message Queue
        switch ( WaitForSingleObject(hMsq, 500) )
        {
             case WAIT_OBJECT_0:
                  // read data from message queue
                  if ( ReadMsgQueue( hMsq, &m_data, sizeof(m_data), &dwRead, INFINITE,
&dwflags) ) {
                       dwDataRecvCount[m_data.id-1]++;
                       dwDataRecvBytes[m_data.id-1] += m_data.dwLen;
                  }
                  printf("\r");
                  for ( int j=0; j<CLIENTS; j++) {
                       printf("ID%d=%3d(%5d) ", j+1, dwDataRecvCount[j],
dwDataRecvBytes[j]);
                  }
                  break;
             case WAIT_TIMEOUT:
                  break;
             case WAIT_FAILED:
                  printf("WAIT_FAILED[%d]\n", GetLastError());
                  break;
             default:
                  break;
        }
        if (bQuit)
             break;
   }

   // close message queue
   CloseMsgQueue(hMsq);

   return 0;
}

DWORD thread_collectdata(LPVOID pnIndex)
{
   int nID = *(int*)pnIndex;
   MSGQUEUEOPTIONS msq;

   msq.dwSize = sizeof(MSGQUEUEOPTIONS);
   msq.bReadAccess = FALSE;                 // Write permission
   msq.dwMaxMessages = 0;                  // Set the max messages number of message
queue to be INFINITE
   msq.dwFlags =  MSGQUEUE_NOPRECOMMIT | MSGQUEUE_ALLOW_BROKEN;
   msq.cbMaxMessage = sizeof(_DATA_t);        // assign to message data structure size

   // create a write only message queue
   HANDLE hTask = CreateMsgQueue(QUEUE_NAME, &msq);

   while (TRUE)
   {
        _DATA_t data;
        data.id = nID;                 // Write ID to identify who I am
        data.pData = NULL;              // not use
        data.dwLen = getRand(nID);       // get random number

        // write data into message queue
        WriteMsgQueue( hTask, &data, sizeof(data), INFINITE, 0);

        // sleep few times
        Sleep( nID * 200 );

        if (bQuit)
             break;
   }
```

```
   // close message queue
   CloseMsgQueue(hTask);

   return 0;
}

int getRand(int nID)
{
   EnterCriticalSection(&cs);

   /* Seed the random-number generator with GetTickCount so that the numbers will be
different every time we run. */
   srand( GetTickCount() );

   /* get a number between 0 and 499 */
   int nResult = rand() % 500;

   LeaveCriticalSection(&cs);

   return nResult;
}


int _tmain(int argc, _TCHAR* argv[])
{
   DWORD dwData[CLIENTS];

   // Initializing the critical section variable
   InitializeCriticalSection(&cs);

   // create the main thread first for waiting data
   CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread_main, NULL,0, NULL);
   Sleep(500);

   for ( int i=0; i<CLIENTS; i++) {
        dwData[i] = i+1;
        // create the client thread
        CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread_collectdata,
&dwData[i],0, NULL);
   }

   // wait a command to close all threads
   WCHAR ch = getwchar();
   bQuit = TRUE;
   DeleteCriticalSection(&cs);

   // make sure all threads were closed
   Sleep(3000);

   return 0;
}
```

A

Frequently Asked Questions

Q: I have an eVB3.0 program. Can I run it on the embedded computer?

A: The old version of Embedded Windows CE programs cannot run on the embedded computer. The target computer is headless with no screen. It may only run applications that are at the console level. To port your console program, paste your source code into a new Visual Basic .Net 2005 console application project. You may encounter warning messages, but the dynamic help screens should help you get through the process.

Q: I have a program under .NET Compact Framework 1.0 (CF1.0). Can I run it on the embedded computer?

A: The target computer pre-installs the version 2.0 engine, which is compatible with version 1.0. Try executing your program on the computer directly. If that does not work, open the project (CF1.0) with Visual Studio 2005 (CF2.0). You may encounter migration messages and code warnings. After making step by step modifications as instructed, your program should be able to run on the embedded computer.

Q: I have a Windows Visual C++ program. Can I run it on the embedded computer?

A: WinCE APIs is a subset of Win32 APIs. Cares must be taken when migrating to it. First, create a Visual Studio 2005 smart device VC project. Add all associated files into the project and rebuild the program. You may see compiling errors, which you should be able to correct by referring to the help pages.

Q: I have a pure C program. Can I run it in the embedded computer?

A: Create an eVC4 project or Visual Studio 2005 Smart Device VC project. Add your C files to your project and rewrite your main function to **winmain()** or **_tmain()** as generated by the tool.

Q: Can I compile C/C++ codes in Visual Studio 2005 other than eVC4?

A: Yes, but you will still need the Moxa C/C++ SDK.

Q: How many concurrent applications can be run on the embedded computer?

A: 32 applications can run concurrently on the embedded computer, each with a 32M RAM limit for Windows CE 5.0.

Q: Are there code examples that show how to capture the function key triggers?

A: Yes, you may find code examples in the programmer's guide.

Q: How can I display a message on the LCM display from my program?

A: You may find code examples in the programmer's guide. The LCM has a display area of 8x16 characters in text mode.

Q: Which version of Compact Framework is installed on the embedded computer?

A: Embedded computers are pre-installed with Compact Framework 2.0. If you have a new compact framework, such as .Net Compact Framework 3.5, you can install it on a Moxa WinCE embedded computer.

Q: What is the Moxa WinCE SDK? What can I do with it?

A: This SDK allows you to write programs in C/C++ that will run on embedded computers. With this SDK installed in the environment of Microsoft Embedded Visual C++ 4.0 or Visual Studio 2005, you can develop WIN32 applications, DLL, and others. In addition, the SDK provides a C library, namely mxdev.lib, which allows control of the LCM, function keys, buzzer and other hardware devices.

Q: When I install the Moxa WinCE SDK, a message appears saying that no tool related to managed code is installed. What is that?

A: That message is for the .NET development environment. Just ignore it and continue with the installation.

Q: Are emulators supported in the SDK?

A: Emulators are not supported in the SDK.