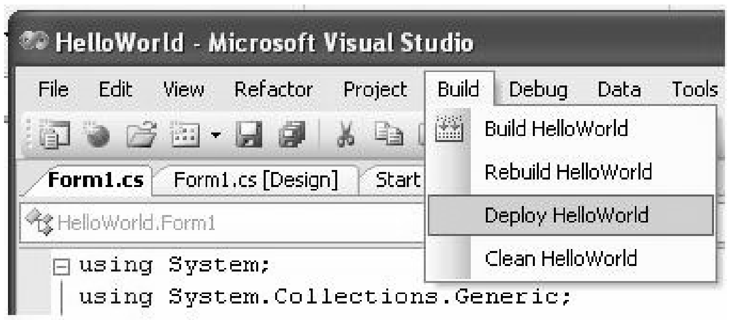
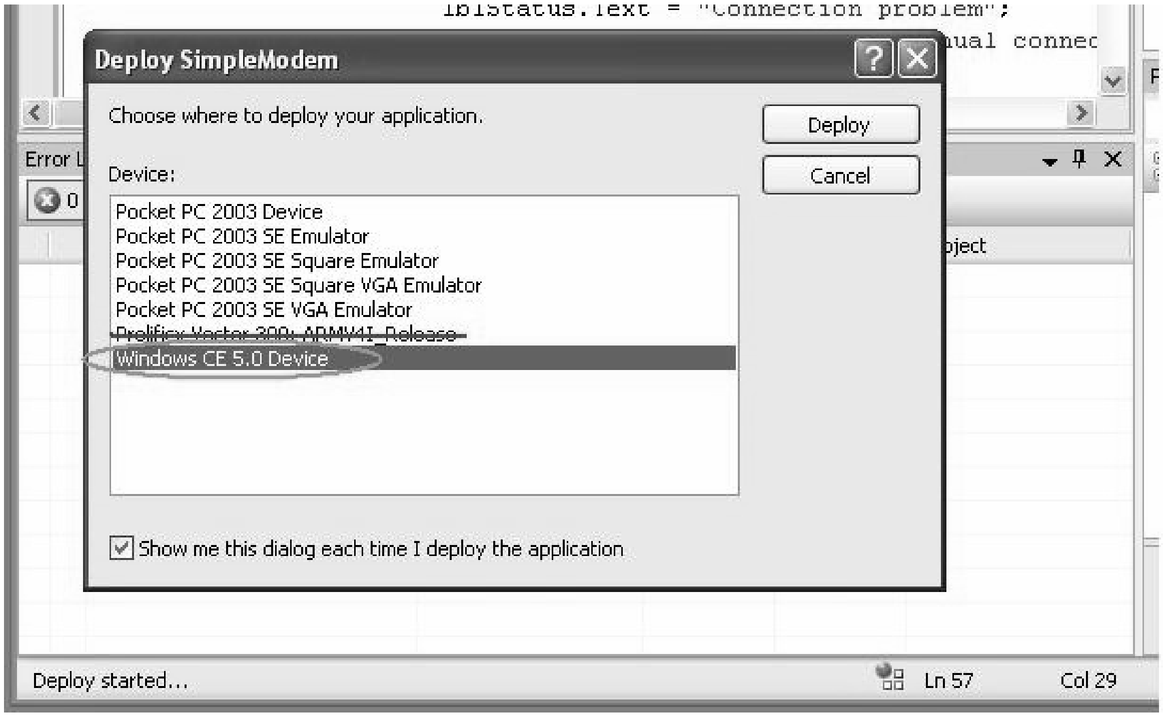
Software Deployment

Now that you've got a project designed and ready to go, you need to "deploy" it to the device. This is as simple as clicking on the build menu, and selecting "Deploy", as shown below.

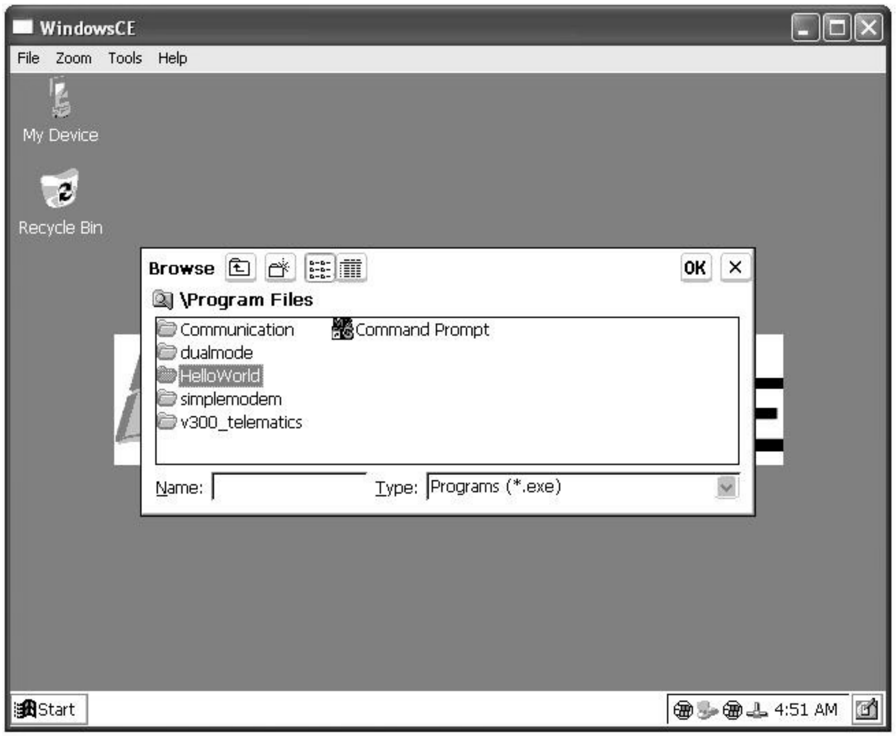

HelloWorld - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools
Form1.cs Form1.cs [Design] Start
HelloWorld.Form1
Build HelloWorld
Rebuild HelloWorld
Deploy HelloWorld
Clean HelloWorld

```
using System;
using System.Collections.Generic;
```

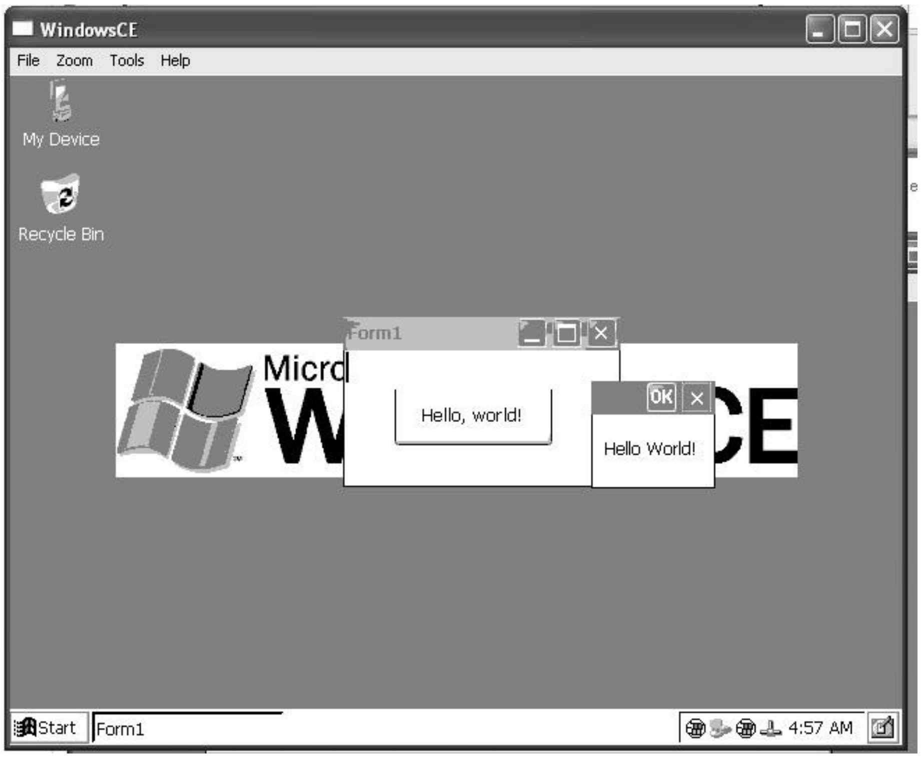
Once you perform that step, Visual Studio will ask you which device to deploy it to. We know it is counterintuitive, but select Windows CE 5.0 device and NOT Prolificx Vector 300, as shown.


Deploy SimpleModem
Choose where to deploy your application.
Device:
Pocket PC 2003 Device
Pocket PC 2003 SE Emulator
Pocket PC 2003 SE Square Emulator
Pocket PC 2003 SE Square VGA Emulator
Pocket PC 2003 SE VGA Emulator
~~Prolificx Vector 300: ARMV4I_Release~~
Windows CE 5.0 Device
Deploy
Cancel
Show me this dialog each time I deploy the application
Deploy started...
Ln 57
Col 29

Once the program is on the device, open up your Remote Display. Press the Start button, hit "Run", and select Browse to navigate the file system. Browse into the Program Files directory, and you'll see a directory created for your application.

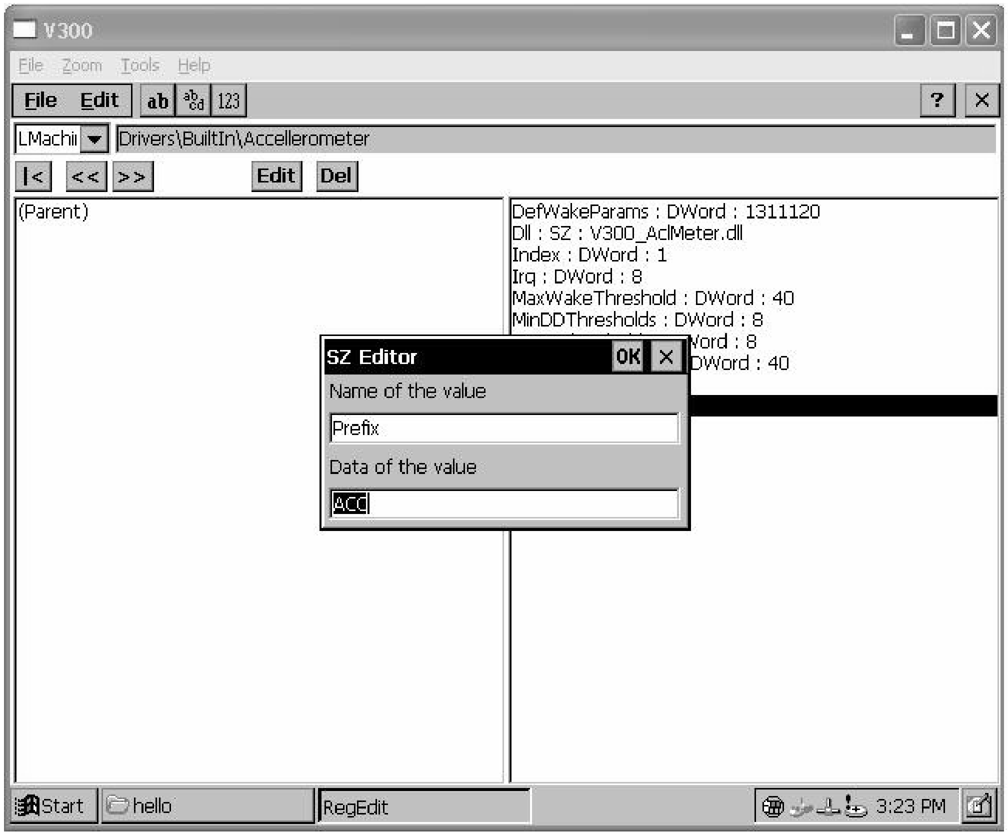


Dive into your application's folder, select the executable, and run it. You should see your first application alive and well on the Vector 300!



Registry Editor

Inevitably you will want to store or view values in the registry of the Vector 300 for various reasons. A simple GUI RegEdit.exe tool is available to download. Put this application onto your Vector 300. All the usual warnings about RegEdit apply here also.



DefWakeParams : DWord : 1311120
Dll : SZ : V300_AclMeter.dll
Index : DWord : 1
Irq : DWord : 8
MaxWakeThreshold : DWord : 40
MinDDThresholds : DWord : 8
ord : 8
DWord : 40

Name of the value
Prefix
Data of the value
ACC

GPRS/CDMA Modem

The Vector 300 is certified and provisioned with a Siemens TC63 quad-band GPRS modem. The TC63 GPRS modem module is controlled via it's AT command interface.

COM Port

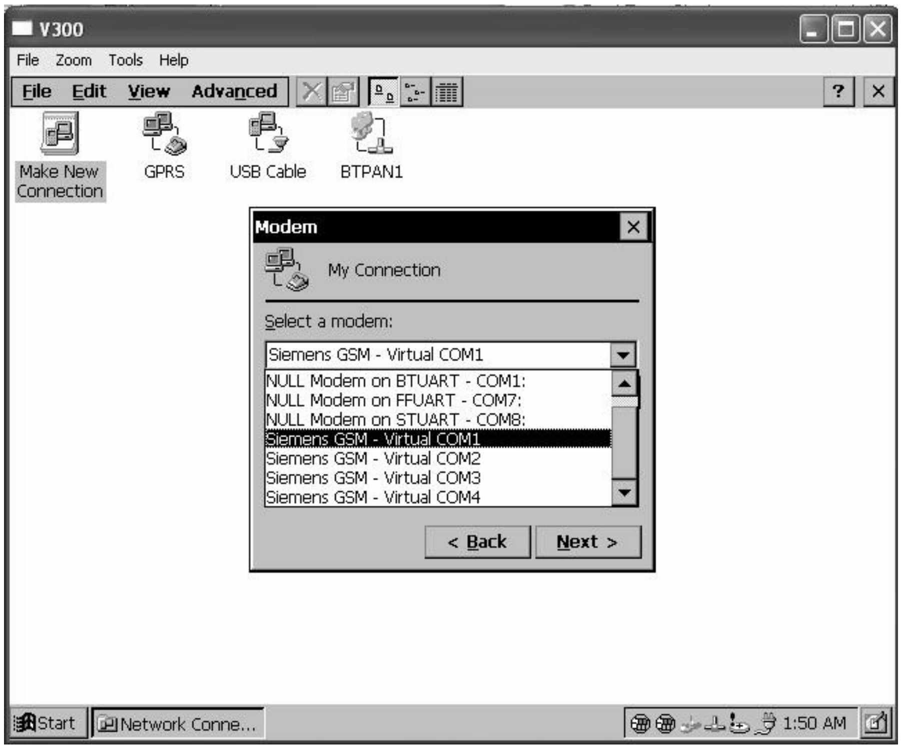
The modem always uses the following information to connect a ***data*** pipe to Virtual COM1. To query the TC63 module via its AT command interface for diagnostics etc, use virtual COM ports 2-4.

For Data:

COM Port	Virtual COM1
Baud Rate	115200
Data bits	8
Parity	None
Stop Bits	1
Flow Control	Hardware

For Control/Diagnostics:

COM Port	Virtual COM2-4
Baud Rate	115200
Data bits	8
Parity	None
Stop Bits	1
Flow Control	Hardware



Modem

My Connection

Select a modem:

Siemens GSM - Virtual COM1

NULL Modem on BTUART - COM1:
NULL Modem on FFUART - COM7:
NULL Modem on STUART - COM8:
Siemens GSM - Virtual COM1
Siemens GSM - Virtual COM2
Siemens GSM - Virtual COM3
Siemens GSM - Virtual COM4

< Back   Next >

Making a Connection

The following material assumes an appropriate antenna has been attached to your Vector 300's GPRS antenna connector.   To make a modem connection to the GPRS network manually, there is a standard procedure that you can follow once you've configured things appropriately. Depending on the carrier you're using, you may not need to do any configuration at all. See the carrier reference below for details.

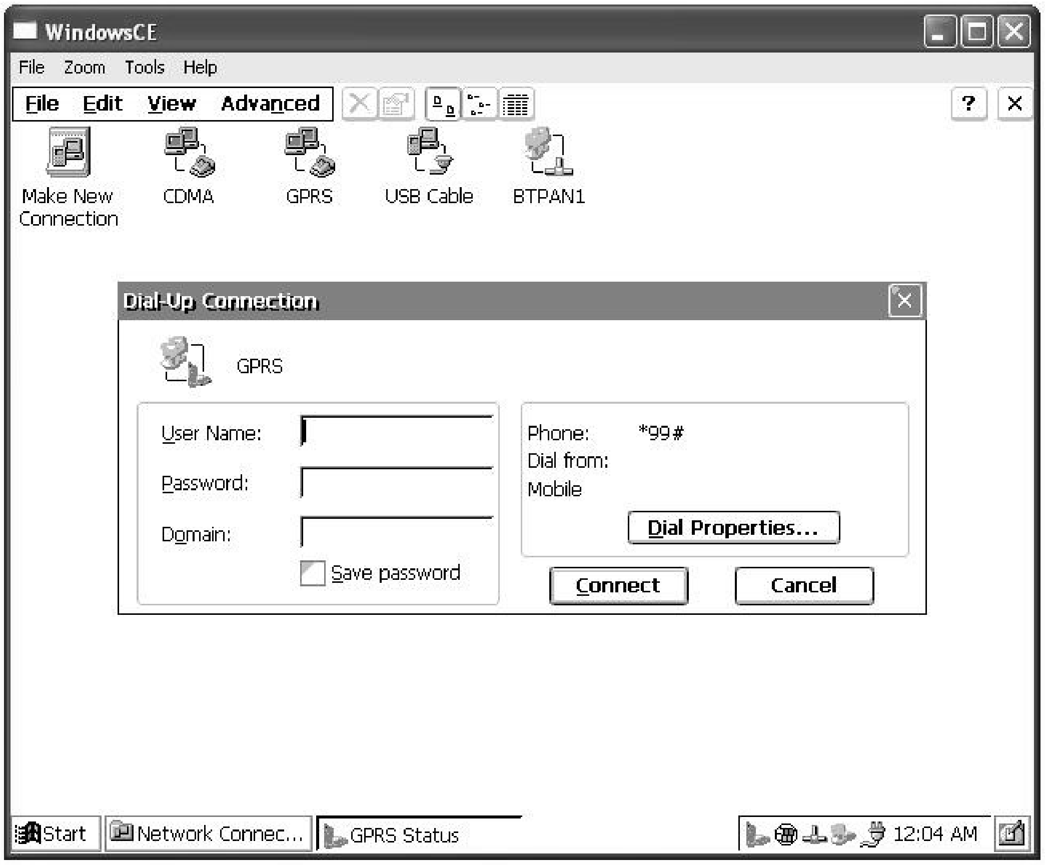
Once you have all your information configured properly, here's the procedure.

General Instructions

Many carriers are configured such that the default settings on the Vector 300 are adequate to make a connection the first time. It is probably best to try these instructions first, and delve into carrier-specific documentation if you experience problems.

1. Start the modem from off state, a cold boot
2. Start up CE remote display
3. Start > Settings > Network & Dialup Connections
4. Hold the mouse over the GPRS connection and press and hold left click button (the circle will pop up indicating a simulated right click)
5. Click 'Connect' on the menu
6. You should see the dialog as shown below if you have a carrier which does not require additional

configuration (i.e., Vodafone). **NOTE: If you are using a carrier which has specific configuration information (as below, such as Cingular or the former AT&T Wireless), you may have different settings, particularly the phone number, username and password.** Refer to the carrier specific documentation to double check.



User Name:

Password:

Domain:

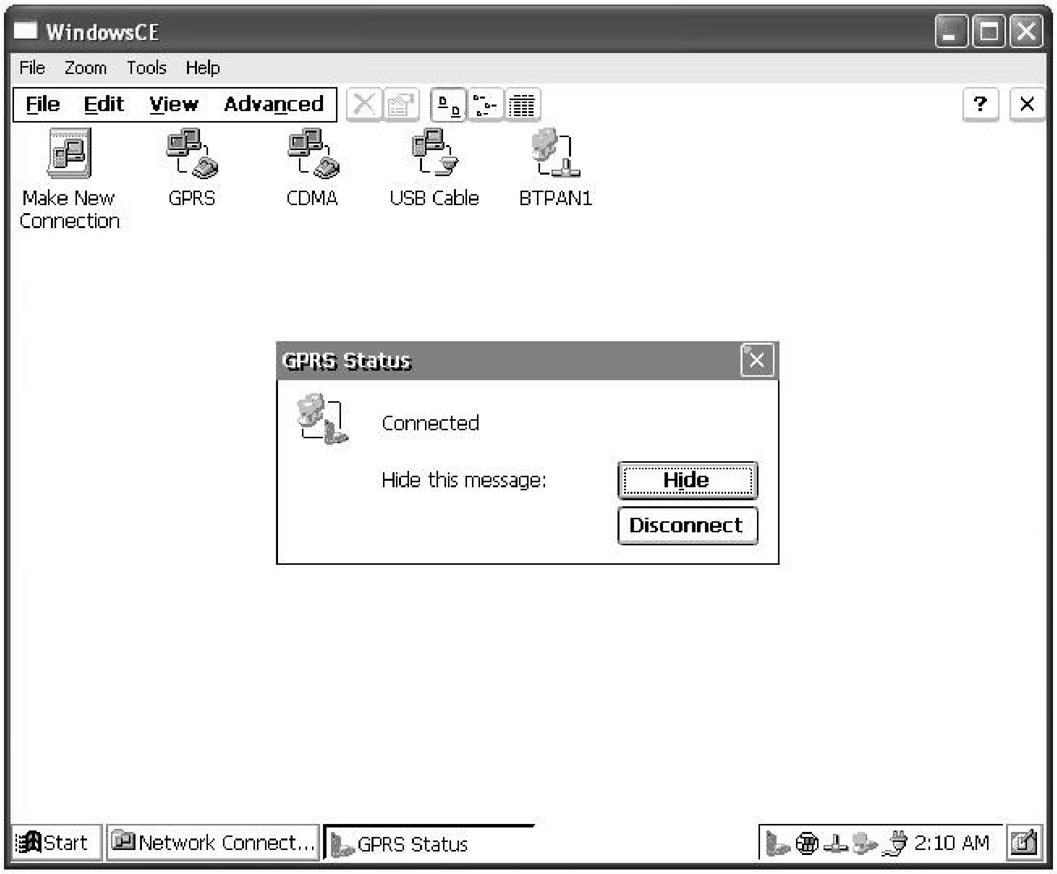
Save password

Phone: *99#

Dial from:

Mobile

7. Click on Connect button above. If you are configured correctly, you should see the following dialog, indicating a successful connection.



GPRS Status

Connected

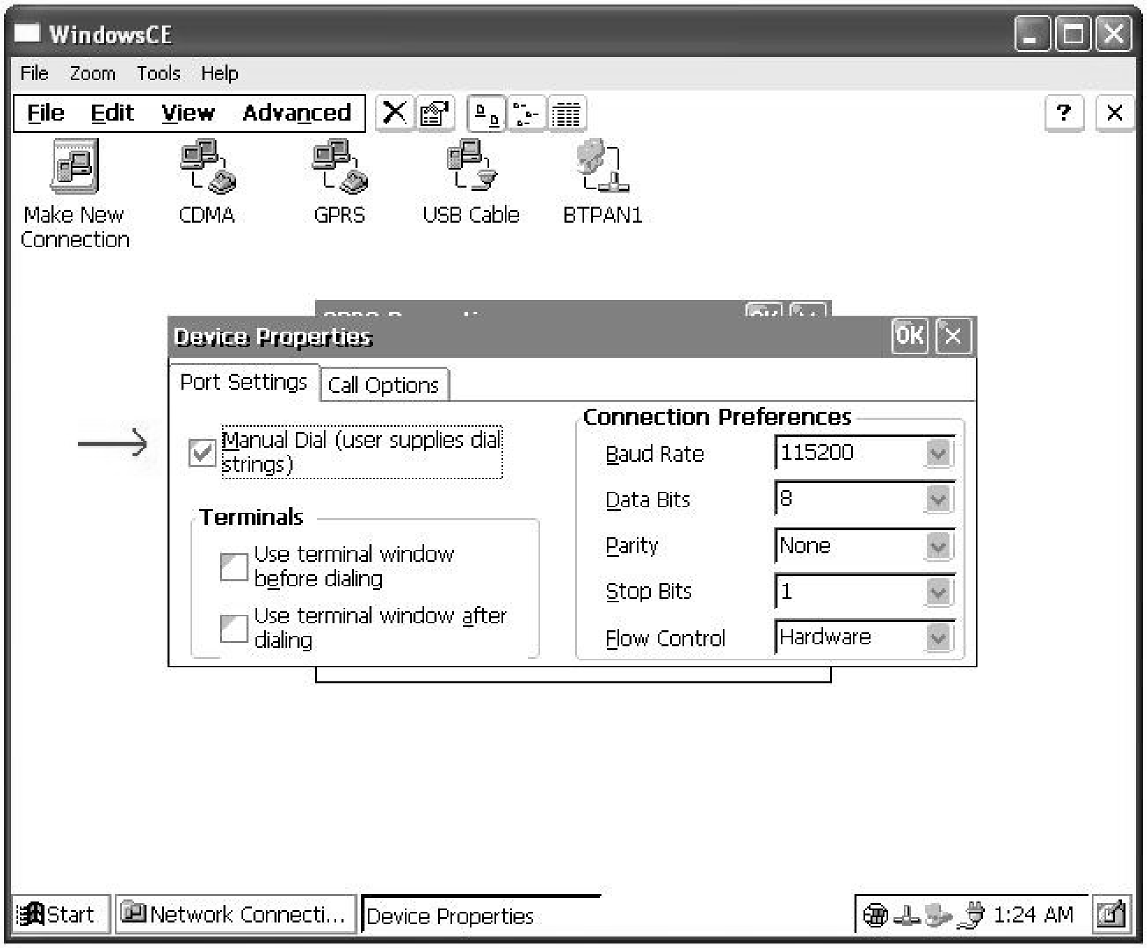
Hide this message:

To verify your connection, follow the instructions below.

Cingular/AT&T (U.S.)

Cingular, a major cellular carrier in the United States, requires what are called "Access Points" to connect to its GPRS network. Cingular has recently acquired AT&T Wireless, formerly also a major carrier in the United States. Although all branded chips now carry the Cingular name, some of them still connect to legacy AT&T systems which have a **DIFFERENT** APN name. Follow the steps below (you only need to do this once) to configure your modem to connect to the Cingular or AT&T network.

1. Start the modem from off state, a cold boot
2. Start up CE remote display
3. Start > Settings > Network & Dialup Connections
4. Hold the mouse over the GPRS connection and press and hold left click button (the circle will pop up indicating a simulated right click)
5. Click 'Properties' on the menu
6. On GPRS Properties dialog, click 'Configure...'
7. Check the "Manual Dial" checkbox.



☑ Manual Dial (user supplies dial strings)

Terminals

☐ Use terminal window before dialing

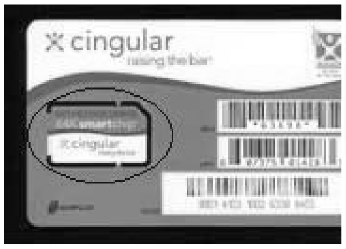
☐ Use terminal window after dialing

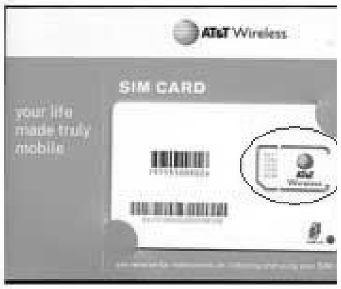
Connection Preferences

Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	Hardware

8. Click OK on the Device Properties and GPRS Properties windows
9. Do the left-click-and-hold on the GPRS connection, and hit Connect. You'll see a dialog box.
10. Make sure all the fields (Username, Password, Domain) are blank and don't contain any data from our previous attempts.
11. Hit the Connect button
12. You're now looking at a direct terminal connection with the modem itself. Now is a good time to identify your SIM card as Cingular or AT&T. Pictures of the SIM cards are below.

Cingular **AT&T**

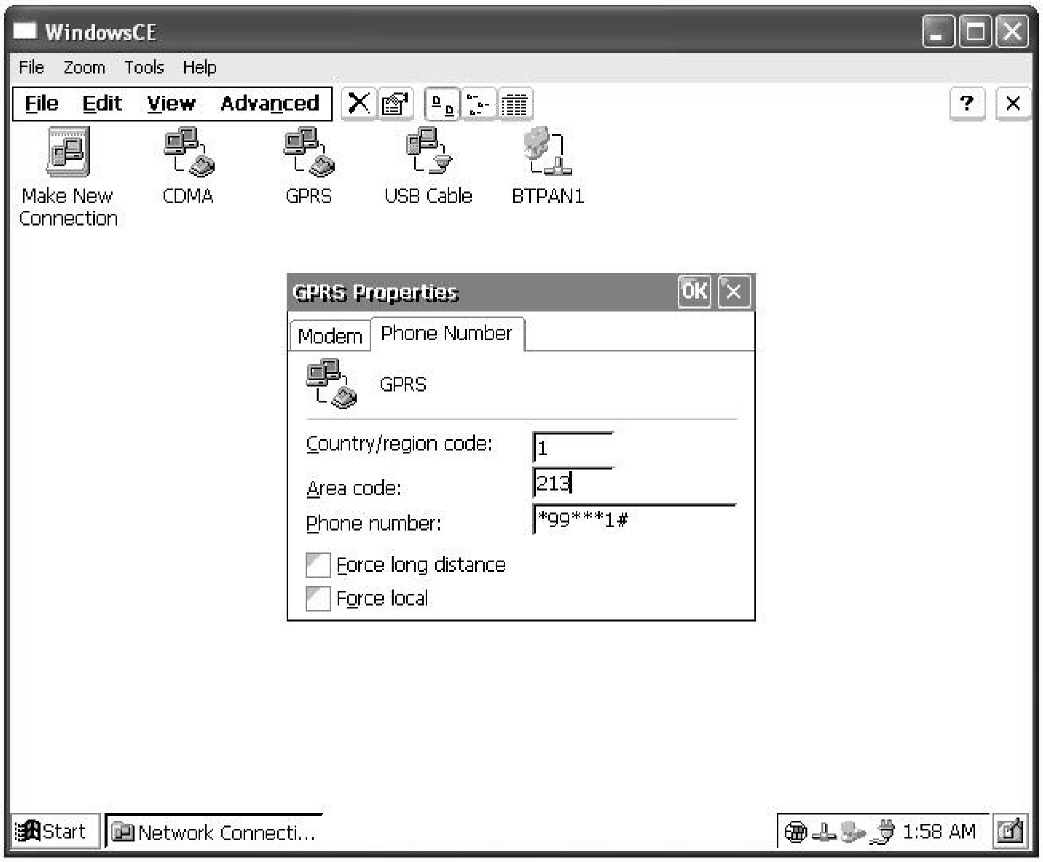




- If you have a Cingular SIM card, enter this command in the terminal window
  - **AT+CGDCONT=1,"IP","isp.cingular"**
- If you have an AT&T SIM card, enter this command in the terminal window
  - **AT+CGDCONT=1,"IP","internet"**

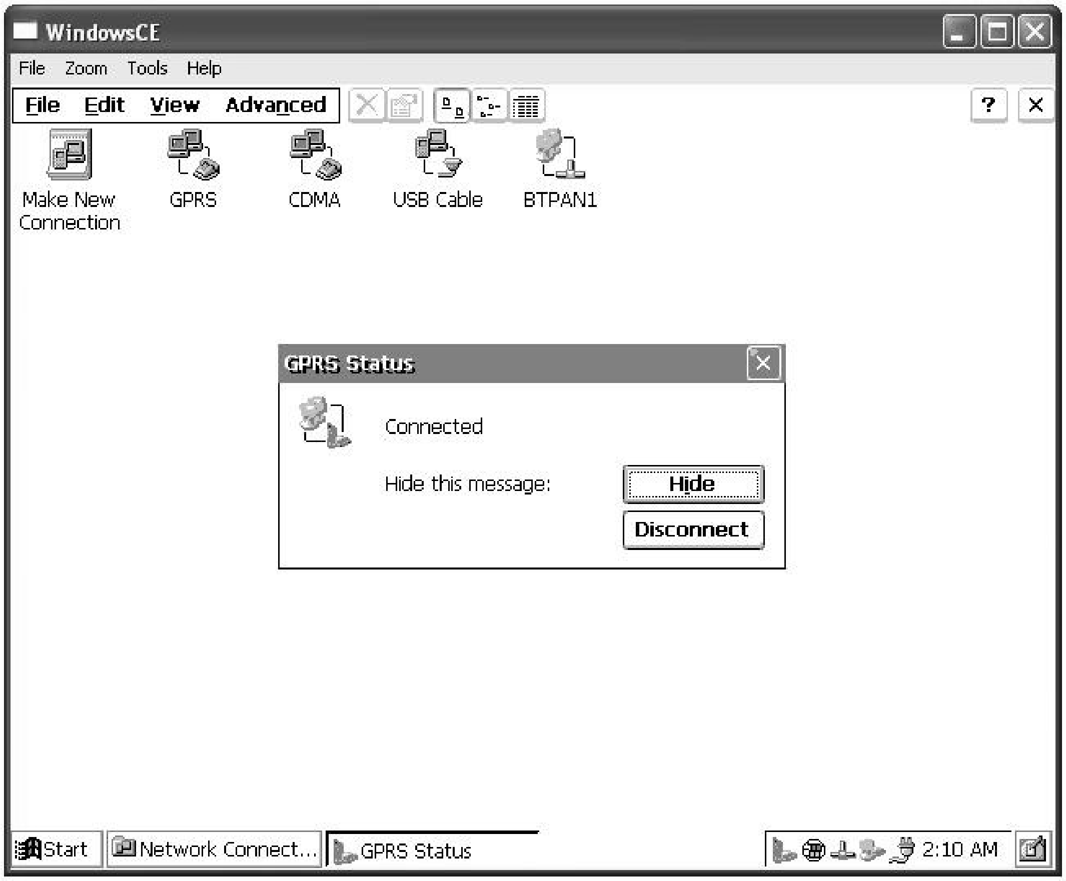
13. You should see the modem report "OK"
14. Now enter: AT&W
15. The modem should again report "OK". Close out the terminal window by hitting the X.
16. Cancel the pending GPRS status connection by hitting cancel and OKing and error messages that come up about not having a carrier.
17. Repeat steps 4 through 8, but this time CLEAR the checkbox on Manual Dial settings so it is unchecked. Click OK on the "Device Properties" window, but do not close the GPRS Properties window. Click on the Phone Number tab and enter ***99***1#** as the phone number. The country code and area code are irrelevant; United States/Los Angeles has been entered as an example only.

    FYI: The normal dial string (default) is *99#. We have adjusted it by adding *****1** between 99 and #. This indicates that we want to use context 1, as specified above in the AT+CGDCONT command. There are up 4 contexts you can define for different carriers, so you could set up both Cingular and AT&T by assigning AT+CGDCONT=1,"IP","isp.cingular" to context 1, and AT+CGDCONT=2,"IP","internet" to context 2. If you had done that, you could use *99***2# to dial the AT&T context as opposed to the Cingular one. The AT&W command writes the command to the modem's FLASH memory so it persists after a power off of the Vector 300.



Country/region code: 1
Area code: 213
Phone number: *99***1#

18. Click OK to close the GPRS Properties Window. Left click and hold the GPRS connection and click Connect. On the connect dialog, follow specific instructions for AT&T vs. Cingular SIM cards:
    - Cingular
        - UserName (all CAPS): ISP@CINGULARGPRS.COM
        - Password (all CAPS): CINGULAR1
        - Domain: (leave blank)
    - AT&T
        - UserName: (leave blank)
        - Password: (leave blank)
        - Domain: (leave blank)
19. Hit connect, and you should see the following, indicating success!



Connected

Hide this message:

To verify your connection, follow the instructions below.

Programatically

Below is sample code for a small GUI based application in C# that programatically connects to the modem, and reports signal strength. You can download the source and application files.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace SimpleModem
{
     public partial class SimpleModem : Form
     {
         //instance of modem class
         Prolificx.V300.Modem m_Modem;

         public SimpleModem()
         {
                InitializeComponent();

                //Initialize modem class;
                m_Modem = new Prolificx.V300.Modem();

                //Update signal strength:
                lblSignal.Text = m_Modem.SignalStrength.ToString();
                //Set visual indicator of signal
                if (m_Modem.SignalStrength > 0)
                     signalBar.Value = m_Modem.SignalStrength;
         }

         private void btnUpdateSignal_Click(object sender, EventArgs e)
         {
                //Update signal strength:
                lblSignal.Text = m_Modem.SignalStrength.ToString();

                //Set visual indicator of signal
                if (m_Modem.SignalStrength > 0)
                     signalBar.Value = m_Modem.SignalStrength;
         }

         private void btnConnect_Click(object sender, EventArgs e)
         {
                //Use button state to determine what user wants
                if (btnConnect.Text == "Connect")
                {
                     //If connecting, check to make sure a connection doesnt already
exist
                     if (m_Modem.HasGPRSConnection)
                     {
                         lblStatus.Text = "Connected to GPRS";
                         btnConnect.Text = "Disconnect";
                         return;
                     }
                     else
                     {
                         if (m_Modem.RASConnect())
                         {
                                lblStatus.Text = "Connected to GPRS";
                                btnConnect.Text = "Disconnect";
                                return;
                         }
                         else
                         {
                                //a Connect failure is usually due to the port
already being open
                                lblStatus.Text = "Connection problem";
                                MessageBox.Show("Do you have a manual connection
already open? If so close it.");
                         }
                     }
                }
                else //User wants to disconnect
                {
                     if (m_Modem.RASDisconnect())
                     {
                         lblStatus.Text = "Disconnected";
                         btnConnect.Text = "Connect";
                     }
                }
         }
     }
}
```

Verifying Connectivity

To verify that your GPRS connection is working properly, go to Start > Run (on the Vector) and type in "cmd". In the terminal window, type in "ipconfig". You should have a an IP issued to you on the Cingular data network via the GPRS modem. In order to ping a host, you'll need to start a "ping –t x.x.x.x" in the command Window, where x.x.x.x is a pingable IP. This will initially *not* connect to the Internet, but unplug the ActiveSync cable, wait 5 seconds, reconnect it, and then you should see replies to your pings. This is due to the routing table configuration which we're currently working on making more user friendly for development purposes; the problem is that when you connect ActiveSync it creates a virtual ethernet adapter that takes precedence over other adapters (the GPRS connection is one of them), so when packets are sent out it chooses the highest precedence adapter, the ActiveSync adapter, instead of the GPRS adapter.

This process will be simplified in a future release of firmware.

Sending a Packet

The .NET sockets collection makes sending a UDP packet very simple. Here is an example of how to send a packet using the simple UdpClient class included in System.Net.Sockets.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;

namespace UDPSample
{
     public partial class Form1 : Form
     {

         public Form1()
         {
                InitializeComponent();
         }

         public String SendGPSPacket(String ip, int port, String latitude, String
longitude)
         {
                String ascii;
                Byte[] data;
                String msg;
                UdpClient client;

                try
                {
                     client = new UdpClient();
                     ascii = "Latitude: " + latitude + ", Longitude: " + longitude;
                     data = Encoding.ASCII.GetBytes(ascii);
                     client.Send(data, data.Length, new
IPEndPoint(IPAddress.Parse(ip), port));
                     msg = "Packet sent!";
                }
                catch (Exception ex)
                {
                     msg = "Unable to send packet: " + ex.ToString();
                }

                return (msg);
         }

         private void button1_Click(object sender, EventArgs e)
         {
         //Replace with your parameters below
                MessageBox.Show(SendGPSPacket("0.0.0.0", 12345, "00.00", "00.00"));
         }
     }
}
```

GPS Receiver

The following material assumes an appropriate antenna has been attached to your Vector 300's GPS antenna connector. The Vector 300 exposes its GPS communications to your host PC through a standard COM port interface. The specifications are as follows:

COM Port	COM8
Baud Rate	38400

Data bits	8
Parity	None
Stop Bits	1
Flow Control	Hardware

The GPS receiver really requires a programmatic interface to make sense of its data. See the example below for details.

Example Program

Below is an example of a simple C# application which receives, processes and displays the GPS data. Prolificx makes use of the PocketGpsLib decoding software, which has an open source license. We have integrated PocketGpsLib into our assembly for your convenience. The example below shows the complete code you'll need to run the application, but there are designer files you need to get it running on your device. The code below is for example only, you should download the source first.

Update: the included source now has a distance calculation function as well.

Download the SimpleGPS example application. Copy “PocketGpsLib2.dll” to the device.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Prolificx;
using Prolificx.V300;
using System.Threading;
using PocketGpsLib;

namespace SimpleGPS
{
     public partial class Form1 : Form
     {
         private GPSHandler m_GPS; //Connects to com port and processes NMEA
sentences
         private System.Threading.Timer m_GPSTimer; //Updates GPS according to
interval
         private const int m_GPSUpdateInterval = 1000; //Defines update interval
         private bool closing = false; //Tells update function that the program is
being closed
                                                 //Used to avoid exceptions

         public Form1()
         {
                InitializeComponent();
         }

         private void Form1_Load(object sender, EventArgs e)
         {
                m_GPS = new GPSHandler(this);
                m_GPS.Start("COM8", 4800);
                m_GPSTimer = new System.Threading.Timer(new TimerCallback(UpdateGPS),
null, m_GPSUpdateInterval,
System.Threading.Timeout.Infinite);

         }

         private void Form1_Closing(object sender, CancelEventArgs e)
         {
                closing = true; //Signal InvokeUpdateGPS function to quit

                lock (m_GPS) //Get a thread synchronized lock on the object to make
sure InvokeUpdateGPS is done with it
                {
                     m_GPS.Stop();
                     m_GPS.Dispose();
                }
         }

         //This is a wrapper function that must be called
         //to "invoke" the function which changes the textbox controls
         //on the form. It must be invoked against the form itself,
         //this simply means that the function will be called by the
         //form's thread. This is a requirement by the .NET runtime
         //in order to avoid thread synchronization issues, it will
         //throw an exception otherwise
         private void UpdateGPS(object unused)
         {
                try
                {
                     this.Invoke(new
EventHandler(InvokeUpdateGPS));
                }
                catch (Exception ex)
                {
                     String dontcare = ex.ToString();
                }
         }

         //This is the invoked function which actually updates the
         //controls on the form
         private void InvokeUpdateGPS(object unused, EventArgs e)
         {
                if (closing)
                     return;

                double tmp;


                try
                {
                     if (!m_GPS.HasGPSFix) //Is a GPS fix available?
                     {
                         this.textBoxFixStatus.Text = "No Fix";
                     }
                     else
                     {
                         //Latitude and longitude can be long decimals, so we trim
them down to 6 decimal places
                         textBoxLatitude.Text =
m_GPS.GPRMC.Position.Latitude.ToString("00.000000");
                         textBoxLongitude.Text =
m_GPS.GPRMC.Position.Longitude.ToString("000.000000");
                         textBoxAltitude.Text = m_GPS.GPGGA.Altitude.ToString();
//meters
                         textBoxKnots.Text = m_GPS.GPRMC.Speed.ToString();
                         tmp = m_GPS.GPRMC.Speed / 1.15077945; //conversion factor
knots->MPH
                         textBoxMPH.Text = tmp.ToString("00.00");
                         tmp = m_GPS.GPRMC.Speed / 1.852; //conversion factor knots-
>KMH
                         textBoxKMH.Text = tmp.ToString("00.00");
                         textBoxNumofSats.Text = m_GPS.GPGSV.SatsInView.ToString();
                         textBoxFixStatus.Text = "Valid Fix";
                     }

                     //schedule timer again
                     m_GPSTimer = new System.Threading.Timer(new
TimerCallback(UpdateGPS), null, m_GPSUpdateInterval,
System.Threading.Timeout.Infinite);
                }
                catch (Exception ex)
                {
                     String dontcare = ex.ToString();
                }
         }
     }
}
```

Accelerometer

The Vector 300 has a built in Accelerometer independent of the GPS module that detects device movement in three dimensions. To access it programatically, use the example code below.

The axes' positions with respect to the V300:

- ☐ The accelerometer's X axis is the longest dimension (length).
- ☐ The accelerometer's Y axis is the short dimension (width).
- ☐ The accelerometer's Z axis is the vertical dimension (height).

In order to run this application copy the “ProlificxCF2.dll” and “V300.dll” to the device.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Prolificx.V300;

namespace AclMeter_Example
{
     public partial class Accelerometer : Form
     {
         public Accelerometer()
         {
                InitializeComponent();
         }
         private Prolificx.V300.MotionSensor Motion;
         private UInt16 prevXAxis = 0, prevYAxis = 0, prevZAxis = 0;

         private void timer1_Tick(object sender, EventArgs e)
         {
                UInt16 diff;
                UInt16 currAxisValue = Motion.ACC_X();
                diff = AxisDiff(currAxisValue, prevXAxis);
                if (diff < progressBar_AxisX.Maximum)
                     this.progressBar_AxisX.Value = (int)diff;
                prevXAxis = currAxisValue;

                currAxisValue = Motion.ACC_Y();
                diff = AxisDiff(currAxisValue, prevYAxis);
                if (diff < progressBar_AxisY.Maximum)
                     this.progressBar_AxisY.Value = (int)diff;
                prevYAxis = currAxisValue;

                currAxisValue = Motion.ACC_Z();
                diff = AxisDiff(currAxisValue, prevZAxis);
                if (diff < progressBar_AxisX.Maximum)
                     this.progressBar_AxisZ.Value = (int)diff;
                prevZAxis = currAxisValue;

                /// check for any motion
                this.checkBox1.Checked = (((this.progressBar_AxisX.Value != 0) ||
                                        (this.progressBar_AxisY.Value != 0) ||
                                        (this.progressBar_AxisZ.Value != 0)) ? true :
false);
                this.AccelPoll.Enabled = true;
         }

         private int ABS(int x)
         {
                return ((x >= 0) ? x : (-x));
         }

         private UInt16 AxisDiff(UInt16 currAxisValue, UInt16 prevAxisValue)
         {
                UInt16 diff;
                diff = (UInt16)(ABS(prevAxisValue - currAxisValue) > 0x1000 ?
            (~(ABS(prevAxisValue - currAxisValue))) : ABS(prevAxisValue -
currAxisValue));
                diff = (UInt16)(diff > 0x1000 ? diff & 0x0fff : diff);
                return (diff);
        }
    }
}
```

Real Time Clock

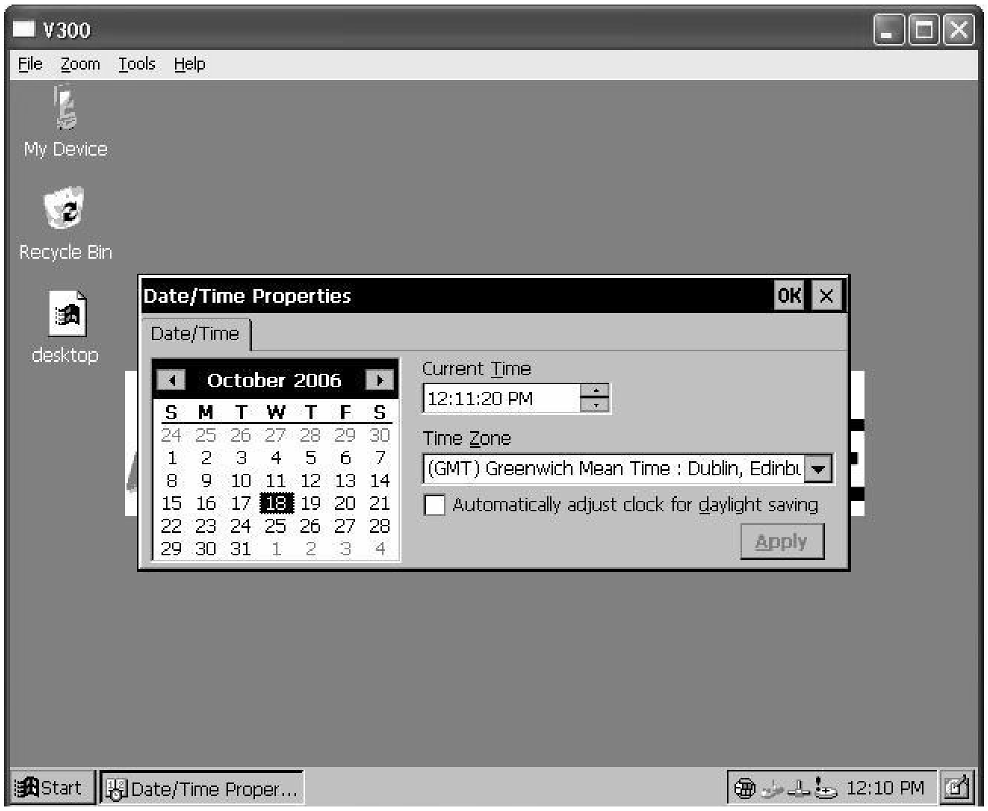
Setting the Real Time Clock

In order to set the current local time and date we can use one of two ways:

☐ The Remote Display GUI interface.
☐ Windows standard API SetLocalTime().

The Remote Display

1. Double click the clock, on the bottom right hand side.
2. Change time and date.
3. Press OK.



Using Windows API

Windows API SetLocalTime( ) enable access to the Real Time Clock.

1. Include header file “Winbase.h”.
2. Fill SYSTEMTIME structure with the required parameters.
3. Call SetLocalTime( ).

BOOL SetLocalTime(const SYSTEMTIME* lpSystemTime);

Parameters:
lpSystemTime - [in] Pointer to a SYSTEMTIME structure that contains the current local date and time.

Return Values:
Nonzero indicates success.
Zero indicates failure.

SYSTEMTIME:

This structure represents a date and time using individual members for the month, day, year, weekday, hour, minute, second, and millisecond.

```
typedef struct _SYSTEMTIME {
WORD wYear;          // Specifies the current year.
WORD wMonth;          // Specifies the current month; January = 1, February = 2, and
so on.
WORD wDayOfWeek;      // This parameter is ignored
WORD wDay;          // Specifies the current day of the month.
WORD wHour;          // Specifies the current hour.
WORD wMinute;          // Specifies the current minute.
WORD wSecond;          // Specifies the current second.
WORD wMilliseconds;      Specifies the current millisecond.
} SYSTEMTIME;
```

```
Example “C” code:
BOOL RTCmodify(void)
{
SYSTEMTIME newTime;

    newTime.wYear=2007;
    newTime.wMonth=8;
    newTime.wDayOfWeek=3;
    newTime.wDay=30;
    newTime.wHour=20;
    newTime.wMinute=7;
    newTime.wSecond=30;
    newTime.wMilliseconds=0;
    if(SetLocalTime(&newTime)==0)
        return FALSE;

return TRUE;
}
```

Bluetooth

The Vector 300 has a Bluetooth module (with internal antenna) on board. To demonstrate how to pass byte streams over Bluetooth, we've created some sample applications. **Generous thanks and credit** are due to **Peter Foot**, author of the **32 Feet Personal Area Networking libraries for .NET**, which are used in these projects.

For futher information on Bluetooth please refer to the following links. These provide more information about registry settings, Networking and connection sharing via Bluetooth.

MSDN Windows CE Bluetooth links and Overview
MSDN Bluetooth Registry Settings.

MS Bluetooth Stack

To demonstrate Bluetooth functionality, you'll need a Vector 300 as well as a development PC with a Bluetooth dongle.

WARNING: The desktop application will NOT work with non-Microsoft Bluetooth stacks.

If you install your dongle and find it is using a 3rd-party driver, such as WIDCOMM, you'll need to install the Microsoft drivers and stack over your 3rd party one. This can be a tricky process, here are some notes:

- First, use the Add/Remove Programs in control panel to remove the WIDCOMM Bluetooth Software.
- Make sure you have Windows XP Service Pack 2, because the Microsoft Bluetooth Stack only comes as part of it
- Check with your manufacturer to get the driver-only release for your dongle. Many manufacturers (i.e. Kensington, which we tested with) don't actually manufacture their devices, they use a third party true hardware manufacturer like MSI. See this article for more information.
- To verify you have the right drivers, look for your dongle in the Windows Device Manager of your host PC (Start > Run > 'devmgmt.msc') to check that the Driver Provider is indeed Microsoft. You know you are on the right track when the device installs and you see Microsoft Bluetooth Enumerator as a new device.

Bluetooth Chat

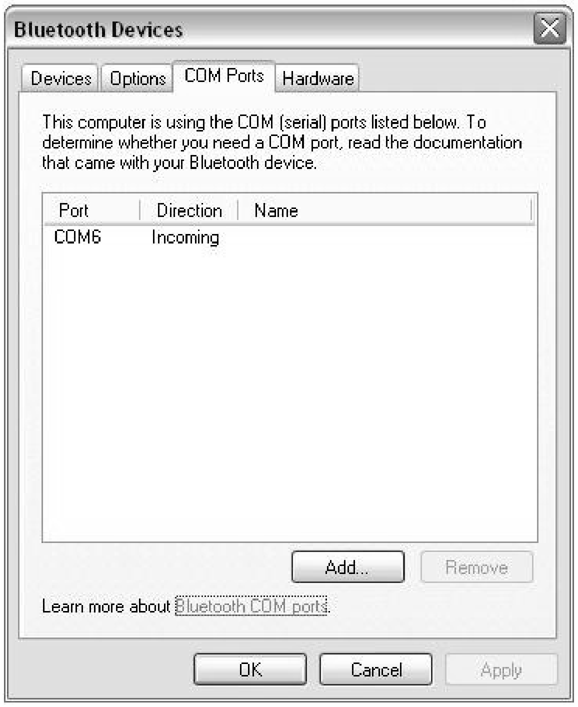
With that out of the way, you're now ready to proceed with making a Bluetooth connection. The zip file here contains two directories -- CEBluetooth and DesktopBluetooth.

Deploy CEBluetooth to your V300, start it up, and hit Discover. Run DesktopBluetooth on your desktop after you've configured your dongle as above, and hit Discover there too. When they've found each other, you'll see the name of your computer (whatever you named it) as the remote device name on the V300, and WindowsCE as the remote device name on the desktop. Send a chat message!

Bluetooth GPS

This example shows you how to use the 32 Feet Personal Area Network libraries along with the Compact Framework 2.0 to open the GPS serial port (COM8), receive its output, and redirect it to a Bluetooth device of your choosing. Make sure you have **read, understood, and implemented** the steps in MS Bluetooth Stack before attempting to do this.

Before you open the application and deploy it to your Vector 300, you need to make sure you have an incoming Bluetooth COM port to listen on. This may need to be manually added. If you've gotten this far, you've already got a dongle installed with the Microsoft Bluetooth Stack. Go to Control Panel, select Bluetooth Devices, and you'll see the dialog shown below. If there aren't any COM ports labeled incoming there, you'll need to add one. Make note of the COM port assigned to you by your machine. With that in hand, open up HyperTerminal by typing in "hypertrm" in Start > Run, and create a new connection to connect to the COM port you just created. You'll notice when you select it that there isn't any place for you to configure port speed or other settings usually associated with a COM port; Bluetooth doesn't need or use these settings. The timer ticking away at the bottom of HyperTerminal indicates that you are connected.


Bluetooth Devices

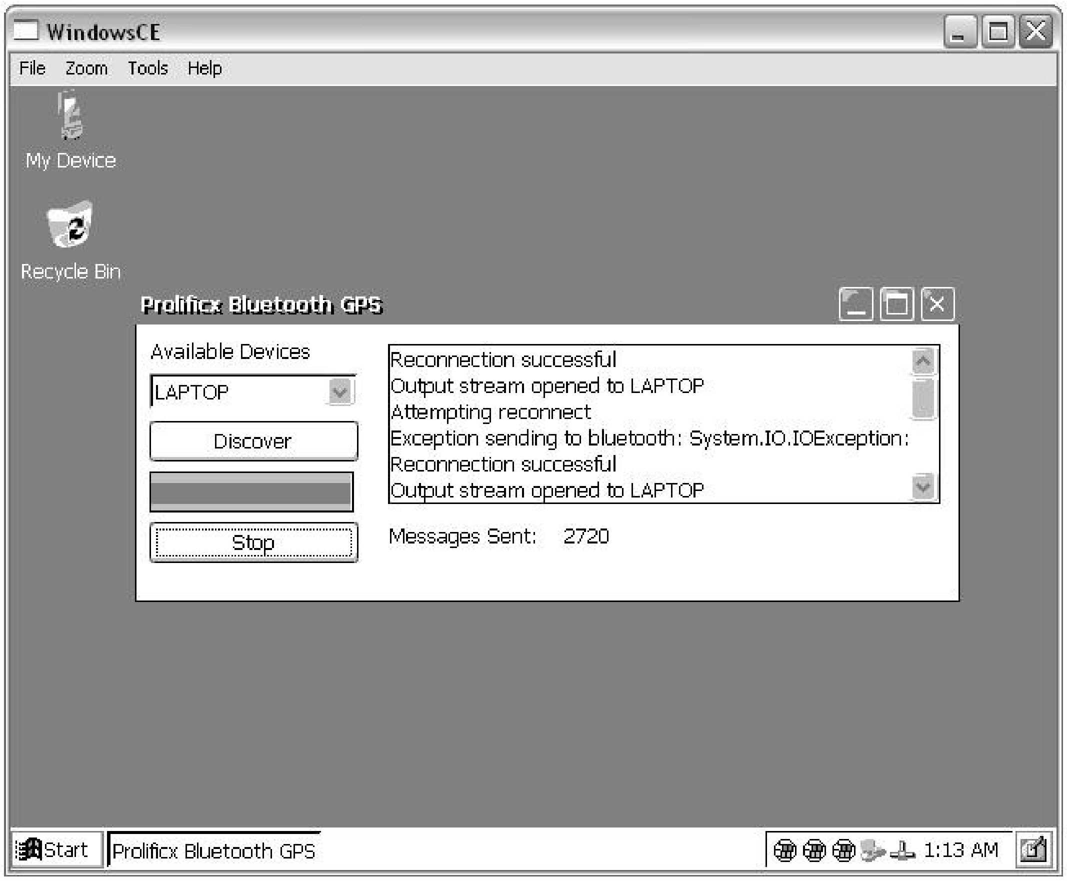
This computer is using the COM (serial) ports listed below. To determine whether you need a COM port, read the documentation that came with your Bluetooth device.

Port	Direction	Name
COM6	Incoming	

Learn more about Bluetooth COM ports.

Now you're ready to deploy the application on to your V300 and start testing!

Download the Bluetooth Serial/GPS Sample. Once you've got it up and running, it should look something like this. You should see NMEA GPS sentences flooding into HyperTerminal on your host PC!



```
Reconnection successful
Output stream opened to LAPTOP
Attempting reconnect
Exception sending to bluetooth: System.IO.IOException:
Reconnection successful
Output stream opened to LAPTOP
```

Messages Sent: 2720

Putting it all together

The following source code is a complete example of how to run a GPS/Telematics processor, complete with modem and GPS port management as well as UDP communications, off the Vector 300. Download the source here. Follow these steps to get the sample working:

1. Download and extract the zip above
2. On a server with a public IP and UDP port open to the internet, open and run the UDPTester.exe application. Make sure you have the .NET framework 1.1 installed or it will not run.
3. Leave remote IP blank, and enter the port of your choice in both Outgoing port and Incoming port fields
4. Hit bind local port, and then Listen for packets from any IP:port. You are now listening for packets
5. Open the project in Visual Studio 2005, edit line 212 in Form1.cs. You will need to supply the IP and port number of your server where the UDPTester application is listening
6. Deploy it to "Windows CE 5.0 device"
7. On the device, browse and run \Program Files\MobileTelematics\MobileTelematics.exe
8. As soon as the program establishes a GPRS connection, unplug the ActiveSync USB cable, and plug it back in after a few seconds to ensure that GPRS becomes the primary route
9. Provided you have a valid GPS fix, you should now see GPS packets about once every second!

Satellite Modem

The Vector 300DM has an internal satellite modem module to provide back up communications via the OrbComm network of LEO satellites when the device is outside of Cellular coverage. The OrbComm network is a low data rate asynchronous communications system designed for small amounts of data in scenarios where latencies can be tolerated. Typical latencies are < 2 minutes.
OrbComm messages are typically delivered as emails to a predefined email address although individual

messages can go to different email addresses as required. OrbComm also provides IP connectivity for accessing messages if email delivery is not desired.

For a more detailed overview of the Orbcomm Network please see the OrbComm General Overview

For detailed information on the messaging services that the OrbComm network provides, please see OrbComm Messaging Services

Antenna selection and mounting is **important** when working with the Orbcomm network. Incorrect selection or installation of the antenna is the most common cause of problems when communicating with the satellites. Also note that many antennas require an adequate ground plane in order to perform to specifications. This application note from OrbComm discusses antenna selection. FCC requirements for the Orbcomm antenna are outlined here.

Note: The satellite modem module included in V300DM is powered independently of all other interfaces. Partly due to this, and partly due to its high power requirements, the satellite modem module **cannot** be used while V300DM is operating from a backup battery. Furthermore, if the vehicle battery voltage dips below 6.5V, the V300DM turns off the satellite modem module, interrupting any ongoing transmissions if necessary to do so.

Satellite Connectivity

When initiating communications with satellites, the modem on the Vector 300 tries to connect to the satellite according to it's configuration settings. Depending on the Vector 300 device application, these settings may need to be modifed by software to set the modem up to search and connect to the required satellites. This application note from Orbcomm discusses Roaming and connection settings.

Modem Connectivity

The satellite modem module connects to the V300 processor via an SPI interface. A driver is provided as part of the operating system that manages all communications with the satellite modem module. This driver implements largely the functionality described in the OrbComm SC Application Program Interface document.   Users do not require details of this interface, as Prolificx provides higher level function calls along with examples of sending and receiving messages.

Sending a Message

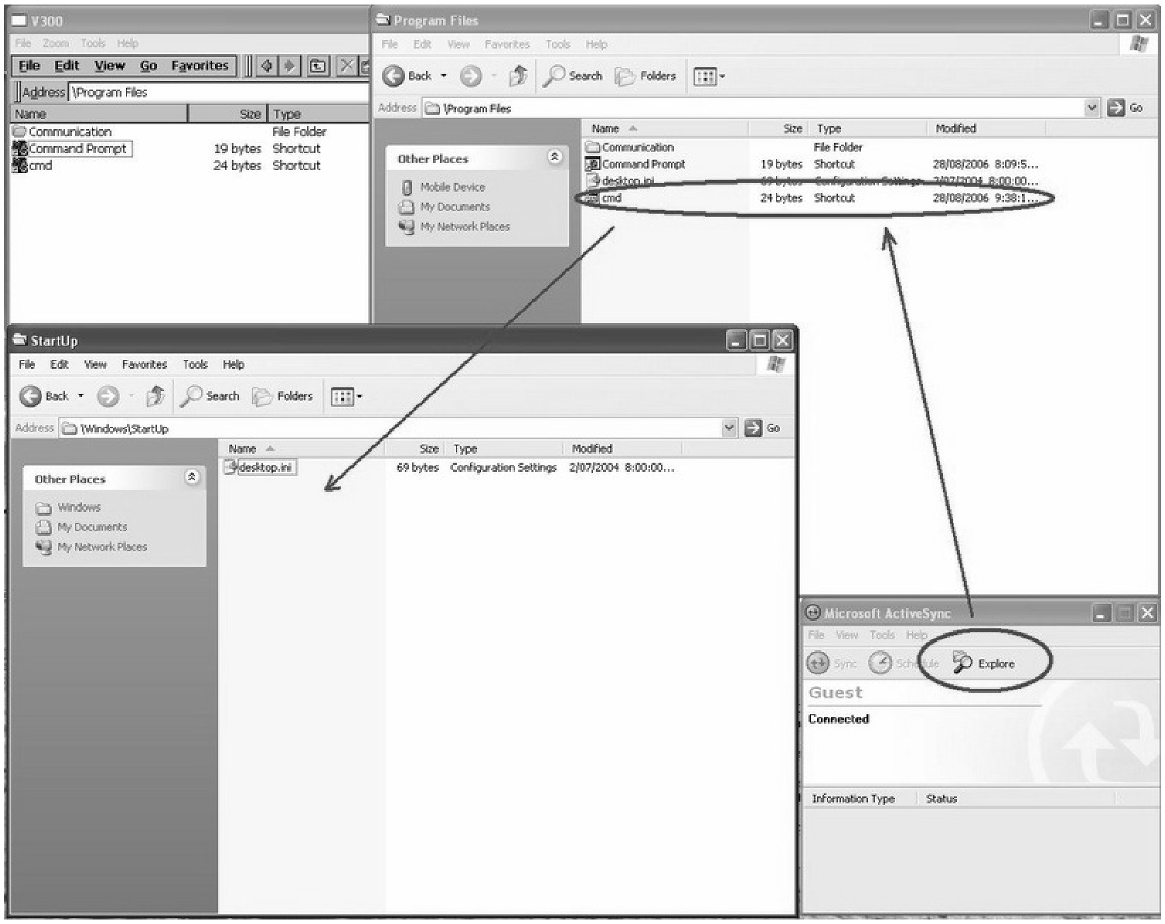
Receiving a Message

Programmers Reference

The Vector 300 may be programmed in either C++ or C# (.NET) environments. This section provides the function definitions and parameters required for accessing on-board peripherals and low level drivers for both languages.

Making Applications Programs AutoRun

To make your programs automatically run when the Vector 300 starts, simply create a shortcut to your program and copy it to the \windows\startup directory.   Moving files and creating shortcuts is done from Explorer available in your host PC ActiveSync window.



Note:
As Windows CE starts applications from the Startup folder running whilst it is still loading some drivers, we suggest putting a delay at the beginning of your application (or otherwise ensuring a delay of 3-5 seconds) before attempting to access the Modem or other peripherals

Port Assignments

The Vector 300 uses the following port assignments:
Virtual COM1 is dedicated to the internal modem data channels (115200,N,8,1)
Virtual COM2-4 are available for GPRS modem control/monitoring channels (115200,N,8,1)
COM1 is for the external serial port or for the J1708. RS232 and J1708 are mutually exclusive. Ref. Registry Settings below.
COM8 is dedicated to the internal GPS (38400,N,8,1)

Bluetooth is available via USB.

I\O Access

Prolificx provides a range of I\O access via the V300.DLL API. This DLL allows us to provide a simple means of reading and writing to the V300 peripherals. It also ensures that the software is easily portable between different hardware products. This DLL provides teh following functionality
- Reading Digital and Analog Inputs
- Writing Digital Outputs
- Accessing the Pulse Counter and Motion Sensor
- Getting version information
- Getting power and system status information
- Setting the power state

C#

Below are code examples / details on accessing the I\O, Pulse counter, Motion Sensor , and power management under C#.
A complete sample demo program is also provided and can be downloaded from here

Code examples (all need to be put in correct sections of program, are grouped together here for simplicity)

```
/******* Digital inputs and outputs     ********/
        // declare Digital IO object
        Prolificx.V300.DigitalIO Digital;
        // Instantiate Digital object
        Digital = new Prolificx.V300.DigitalIO();
        // Set relay 1 output high (outputs are RELAY_1... 4)
        Digital.Write((ushort)IO_PINS.RELAY_1,true);
        // Set relay 1 output low
        Digital.Write((ushort)IO_PINS.RELAY_1,false);

        // read Input 1 state into IOLevel - inptus are 1-5
        Digital.Read((ushort)IO_PINS.DIN_1,out bool IOLevel);
        // read input state of pulse counter- this can be read as a digital I\O or
cntr
        Digital.Read((ushort)IO_PINS.DIN_HS,out bool IOLevel);

/******* Pulse Counter    *******/
         // declare Pulse Counter object
         Prolificx.V300.PulseCounter PulseCnt;
         // Instantiate Pulse counter
         PulseCnt = new Prolificx.V300.PulseCounter();
         PulseCnt.ResetPulseCount();
         // Start Pulse counter - returns True if pulse counter started, false if
exception
         bool PulseCnt.StartPulseCount();
         // Stop PulseCounter - returns True if pulse counter stopped, false if
exception
         bool PulseCnt.StopPulseCount();
         // reads current value of pulse counter in pValue, pOverFlow set if overflow
has occured (Operation of overflow to be confirmed)
         bool PulseCnt.ReadPulseCount(out UInt32 pValue, out Byte pOverflow);


/******* Power management *******/
/******* Control Shutdown and get power and backup battery states *******/
         // declare Power Management object
         private static Prolificx.V300.PowerManagement Power;
         // Instantiate Powermanagement object
         Power = new Prolificx.V300.PowerManagement();
         // Get backup battery voltage and % full
         bool Power.BatteryPowerState(out UInt16 Voltage, out UInt16 Percentage)
;
         // Get Main power voltage and % (% not relevant as not running from battery
normally)
         bool Power.SupplyPowerState(out UInt16 Voltage, out UInt16 Percentage);

         // Put V300 into low power mode - will wake up with Ignition changing from
low to high
         Power.SetPowerState("suspend");

/******* Analog Input *******/
         // declare Analog object
         private Prolificx.V300.ADC Analog;
         // Instantiate Analog Object
         Analog = new Prolificx.V300.ADC();
         // Declare variable for result
         UInt16 Value = 0;
         // Call Analog read functio to get value
         Analog.Read(out Value);

/******* Motion Sensor *******/
// Additional functionality will be aded to this in dues course to support waking up
// from movement and the ability to enable or disable this wakeup
         // Declare motion sensor object
         Prolificx.V300.MotionSensor Motion;
         // Instantiate Motion sensor
         Motion = new Prolificx.V300.MotionSensor();
         // Call function to see motion
         bool MotionDetected = Motion.HasMoved();

/******** IO enumerations ***************/
         enum IO_PINS
        {
                DIN_HS = 1,
                DIN_1,
                DIN_2,
                DIN_3,
                DIN_4,
                DIN_5,
                RELAY_1,
                RELAY_2,
                RELAY_3,
                RELAY_4,
                ANALOG_IN1 = 13
    };
```

C++

```
Input / Output Functions (See below for Enumerations )
                     BOOL ReadPinLevel(DWORD Pin, PBOOL pLevel);
    BOOL ReadAnalogLevel(PWORD pValue);
    BOOL WritePinLevel(DWORD Pin, BOOL Level);

Pulse counter functions
    BOOL ResetPulseCount();
    BOOL StartPulseCount();
    BOOL StopPulseCount();
    BOOL ReadPulseCount(DWORD *pValue, BYTE *pOverflow);

Motion Sensor
    BOOL IsMotionDetected( BOOL* pDetected, BOOL fClearDetectedMotion =TRUE);

Examples
    WritePinLevel(RELAY_1, TRUE);     // turn on Relay 1

enum IO_PINS
{
    DIN_HS = 1,
    DIN_1,
    DIN_2,
    DIN_3,
    DIN_4,
    DIN_5,
    RELAY_1,
    RELAY_2,
    RELAY_3,
    RELAY_4,
    ANALOG_IN1 = 13
};
```

CAN bus

Prolificx provides low level access to the Vector 300's CAN controller (Microchip MCP2515) via a driver built into the operating system. Prolificx is currently developing high level drivers to access CAN based vehicle protocols including ISO15765, J1939 and FMS. Expert knowledge is required to develop CAN protocols. Using the Prolificx developed protocols is strongly recommended. Contact Prolificx for availability details.

An example program for simple ISO15765 access can be downloaded here, however this is a simplistic example solely meant to illustrate communicating to the CAN bus via the CAN controller.

Registry Access

Registry manipulation can be done easily from your code. To demonstrate how simple it is to do this, we've put together a sample application. Really what it boils down to is two simple functions:

```
private String RegistryGet(String key)
{
      String s;
      try
      {
          s = Microsoft.Win32.Registry.GetValue(
          "HKEY_CURRENT_USER\\Software\\Prolificx", key,"").ToString();
      }
      catch (Exception ex)
      {
          String dontcare = ex.ToString();
          s = "";
      }
      return (s);
}

private void RegistrySet(String key, String value)
{

      Registry.SetValue("HKEY_CURRENT_USER\\Software\\Prolificx",key,value);
      Microsoft.Win32.RegistryKey target =
Microsoft.Win32.Registry.CurrentUser.OpenSubKey("SOFTWARE\\Prolificx",true);
      target.SetValue(key, value);
      target.Flush();
      target.Close();
}
```

You can find a simple demo application of how to use this code here:

Registry Settings

Registry Setting can be done using the Registry Editor or through your code using the Windows standard API as demonstrated here.

J1708

The J1708 shares a serial port with the RS232 interface. These two interfaces **cannot** operate concurrently. In order to disable the RS232 driver and enable the J1708 port, change the following registry entry to a value of 1: `HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Serial2\EnableJ1708`

Note this value is only loaded when the device boots, so any changes only take effect **after** the V300 has been rebooted

USB

When requiring larger numbers of devices on the USB Host port, e.g. Hubs or more than one external device, it may be necessary to increase the amount of memory space allocated to the driver, as set by a registry entry.

The current default Page Size allocation is 64K (this default may be increased on subsequent OS builds) which will support typically one external device. To increase the default setting create a new DWORD key PhysicalPageSize in the registry folder `[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\OHCI]` and set it an appropriate value. This key sets aside a common memory pool for the USB driver (size in bytes).

Setting this PhysicalPageSize key to 100000 should support two external devices. Raising it to 1000000 enabled us to test with 4 USB hubs daisy chained to two keyboards, two mice, and three USB memory sticks on a 5m USB active repeater cable.

Specifications

Vector 300 General

☐ Enclosure Dimensions 140 x 89 x 45 mm
☐ Weight 192 to 412 gm, depending on options
☐ Enclosure Flammability: UL V0
☐ Operating Temperature: Full function from -30°C to +70°C

- Capabilities are reduced down to -40°C and up to +85°C.
- Battery function with large battery option allows for approximately 15 minutes of GPRS Class 10 transmission at -25°C.
- Battery charging is disabled below 0°C. Rechargeable battery capacity below -25°C is impaired.
- Storage Temperature: -40°C to +85°C.

☐ Power Supply 9 to 32 V DC main power input (internally monitored) to suit cars and trucks with nominal 12/24V battery systems.**NOTE:** Current drawn depends on options and battery voltage. 12V battery assumed for the following:

- Operational current:
  - Standard Full Operation - 200-400mA, depending on hardware configuration
  - Worst case Peak, with GPRS Class 10 but no satellite modem - 1.0A
  - Worst case Peak, with both GPRS Class 10 and the satellite modem module transmitting simultaneously - 3.0A
- Sleep mode: 10 to 20 mA.

Applicable to standard V300-336-GX model: Contact Prolificx for variants.
☐ 32-bit CPU 312MHz
☐ RAM: 64MB clocked at 104MHz
☐ Flash ROM: 32MB (Upgrade to 64MB available)
☐ MMC/SD Card interface
☐ 20 channel GPS receiver

- 20 channel operation, 1Hz update rate
- Positional Accuracy
  - Autonomous = 10 m
  - WAAS / EGNOS < 5 m
  - Beacon (DGPS) < 2.5 m
- Velocity < 0.1 m/s
- Time Accuracy: Within 1μs of GPS time
- Time to First Position
  - GSM/3GPP: < 20s average
  - CDMA: < 16s average
  - Hot start: < 18s average
  - Hot start open sky: < 1s average
  - Cold start: < 42s average
- Sensitivity: 15 dBHz except Cold start: 30dBHz
- Dynamic Conditions
  - Altitude: 18,000m max
  - Velocity: < 515 m/s max
  - Acceleration: 4g max
  - Jerk: 20 m/s2 max
  - NMEA 38400 bps, 8N1, messaging: GLL, GGA, RMC, GSV, GSA

☐ Mobile Connectivity

- Siemens TC63 for Europe GSM/GPRS 900/1800MHz and US GSM/GPRS 850/1900MHz

☐ 1Mbps CAN (2.0) Interface

☐ Bluetooth V1.2 Class 2 interface with internal antenna
☐ USB client port for Active Sync
☐ USB host port with 5V/100mA max. power feed
☐ Up to six protected Digital Inputs, one Analog Input
☐ Four protected 250mA low-side Relay Driver Output Lines
☐ Integrated real time clock
☐ Remotely upgradeable firmware image
☐ Standard 150mAhr rechargeable Lithium Polymer battery
☐ Visual Indicators: 1 x Red LED (Power Manager), 2 x Green LEDs (OS indicator, Cellular modem activity)
☐ Integrated audio controller with one Lineout standard (see Option Configurations below)

Optional Configurations

The following Options must be stipulated at time of ordering:

- ORBCOMM satellite modem receiver (V300DM option)
- RS-232 serial port*
- RS-485 serial port*
- J1708 serial port*
- Additional audio inputs and outputs (at the expense of Digital Inputs)

* **NOTE:** These options are mutually exclusive. Configuration options are clarified in the IO Connector table here

Optional Accessories

☐ Rechargeable 1500mAhr Lithium-Polymer battery
☐ GSM/GPRS antenna module
☐ GPS active antenna
☐ Approved Orbcomm satellite modem antenna
☐ AC Adaptor
☐ 12V DC cigarette lighter car adaptor

Antenna

GSM Antenna

Environmental	
Temperature	-20°C to +85°C
Protection class	IP66
Mechanical	
Cable type and length	RG174, <3m
Connector	FAKRA Plug, code D
Electrical	
Frequency ranges	GSM850: 824MHz to 894MHz GSM900: 880MHz to 960MHz GSM1800: 1710MHz to 1880MHz GSM1900: 1850MHz to 1990MHz
VSWR	<2.0:1
Radiation pattern	Omni-directional
Gain	<4.4dBi in 1800MHz and 1900MHz bands, <2.98dBi in 850MHz and 900MHz bands
Radiation efficiency	>50%
Impedance	50Ω
Polarisation	Linear - vertical

GPS Antenna

Environmental	
Temperature	-20°C to +85°C
Protection class	IP66
Mechanical	

Cable type and length	RG174, <3m
Connector	FAKRA Plug, code C
Electrical	
Frequency range	1,57542 GHz ± 1.023 MHz, (L1-band)
VSWR	<2.0:1
Gain	>2dBi
Impedance	50Ω
Polarisation	Right hand circular
Amplification	>25dB
Amplifier voltage requirement	<3V DC
Amplifier current requirement	<50mA
Amplifier noise figure (50Ω)	<2.5dB

Orbcomm Satellite Modem Antenna

Environmental	
Temperature	-20°C to +85°C
Protection class	IP66
Mechanical	
Cable type and length	RG174 or RG58, <3m
Connector	FAKRA Plug, code H
Electrical	
Frequency ranges	138MHz to 150MHz
VSWR	<2.0:1
Radiation pattern	Omni-directional
Gain	<6dBi
Radiation efficiency	>50%
Impedance	50Ω
Polarisation	Linear - vertical
Type	¼ λ or ½ λ whip