Lab 11.4: Adding WinSock Capabilities

Objectives

After completing this lab, you will be able to create MFC applications that use the WinSock classes to:

- Create a connectionless (datagram) socket to receive data.
- Create a connectionless (datagram) socket to transmit data.

Prerequisites

You should be able to create MFC applications that invoke and use modal dialog boxes.

You should have completed the section in Chapter 11 titled "Using the WinSock Classes."

Lab Setup

To run the solutions to this lab, you invoke two cooperating applications.

To run the GPSRC solution, which sends packets to GPSNK, click this icon.

To run the GPSNK solution, which receives packets from GPSRC, click this icon.

This demonstration shows what you will accomplish during the lab.

Estimated time to complete this lab: **30 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Creating a Sink Socket Application

In this exercise, you will complete the baseline application GPSNK, which simulates a base station. In GPSNK, you will create a datagram "sink" socket, one that only receives packets. You complete the baseline application for the "source" socket, GPSRC in Exercise 2.

Exercise 2: Creating a Source Socket Application

In this exercise, you will complete the baseline application GPSRC, which simulates a satellite/vehicle/wireless-link subsystem. In GPSRC, you will create a a datagram "source" socket, that is, a socket that only sends packets. It supplies data to the GPSNK application that you create in Exercise 1.

Copy the contents of \Labs\C11\Lab04\Baseline to your working directory.

The completed code for these exercises is in \Labs\C11\Lab04\Xxx, where Xxx is the exercise number.

Exercise 1: Creating a Sink Socket Application

The purpose of this lab is to use the MFC WinSock classes to create two applications that implement a transmitter/receiver socket pair to simulate a vehicle location system based on the Global Positioning System.

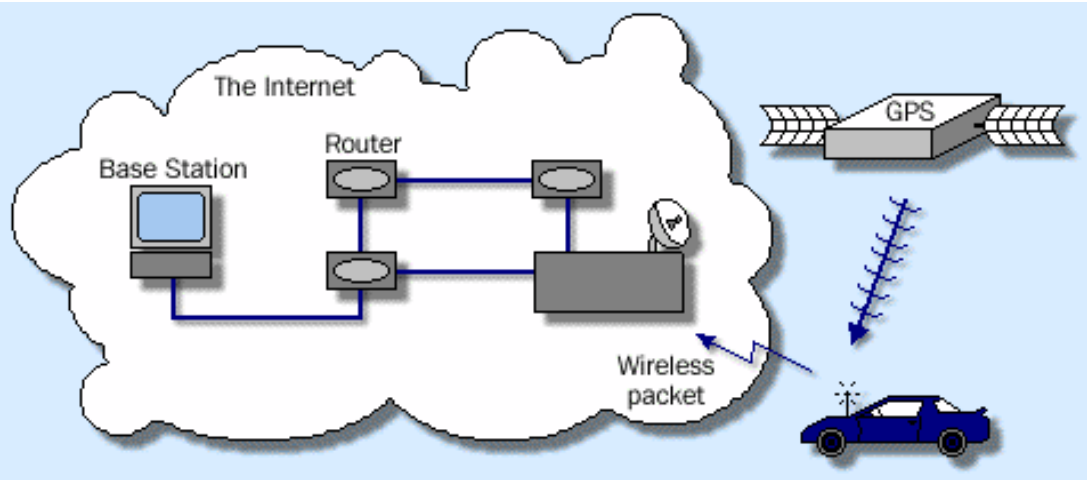
You will simulate a satellite/vehicle/wireless-link subsystem with a datagram "source" socket, that is, a socket that only sends packets. The base station will be simulated by a datagram "sink" socket, one that only receives packets.

In this exercise, you will complete the baseline application GPSNK, which simulates the base station described in the lab scenario. In GPSNK, you will create a datagram "sink" socket, one that only receives packets. You complete the baseline application for the "source" socket, GPSRC in Exercise 2.

About the Two Baseline Applications

Many businesses have fleets of vehicles that must be tracked in real-time in order to run the business efficiently. Combining the Internet's data communication capabilities with the Global Positioning System's (GPS) location capabilities creates the potential for powerful solutions in this area.

GPS consists of 24 satellites that circle the Earth, transmitting positional and timing information to GPS receivers. Using triangulation, GPS receivers can determine their own location, and the vehicle's, to within 100 meters. This information can be encapsulated within a datagram and sent, initially via a wireless Internet transmission device, to the company's base station. The base station uses the information received from the location meter to track the vehicle. This illustration depicts the scenario that the two applications created in Lab 11.4 are designed to address.



You will simulate the satellite/vehicle/wireless-link subsystem with a datagram "source" socket, that is, a socket that only sends packets. The base station will be simulated by a datagram "sink" socket, one that only receives packets.

A timer within the source socket application will periodically trigger, causing it to send a packet to the base station on a preselected port. The packet will contain a longitude/latitude point as an ASCII string. The receiver socket, listening on the preselected port, will receive the packet and then extract the positional information and display it to the user.

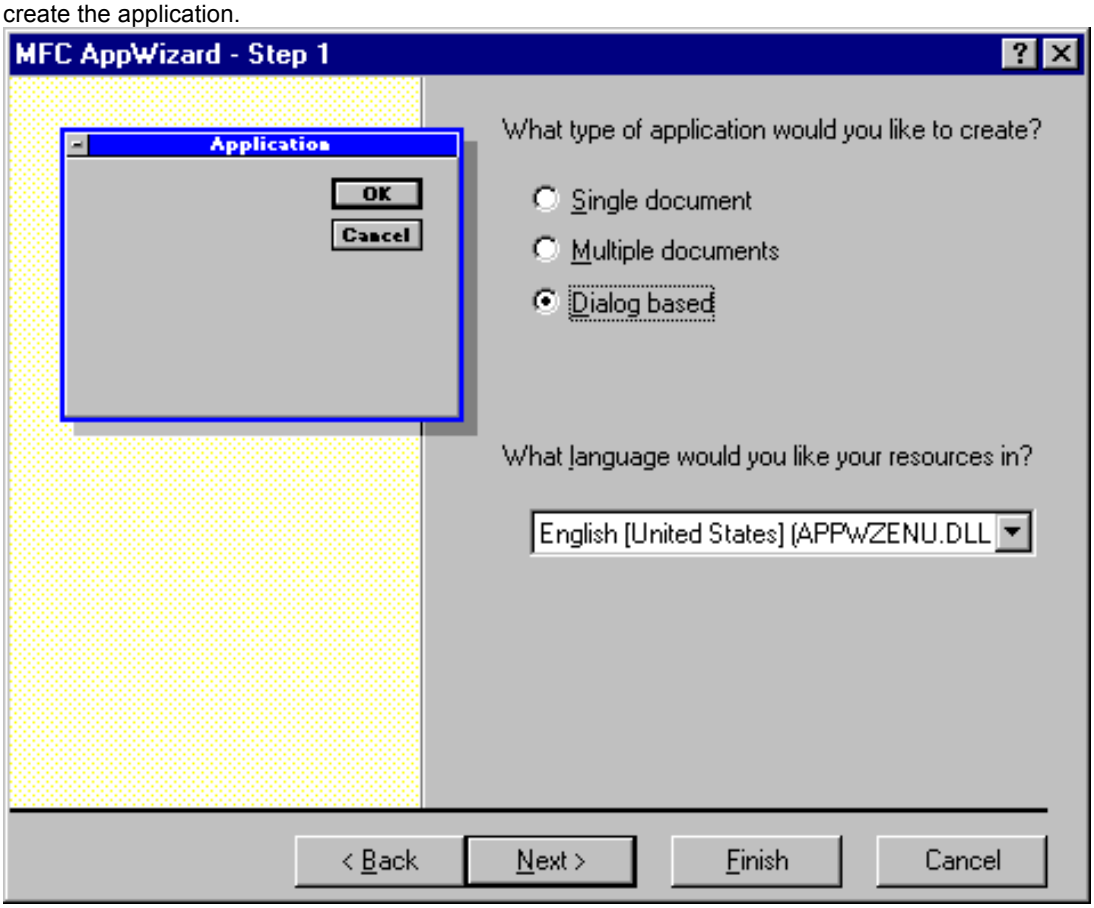
Implementing the solution using a datagram socket rather then a streaming sockets is a design decision. This is a good choice considering that:

1. The loss of a packet is not catastrophic.
2. Tracking dozens of such vehicles precludes the overhead involved with the use of a connection-oriented socket.

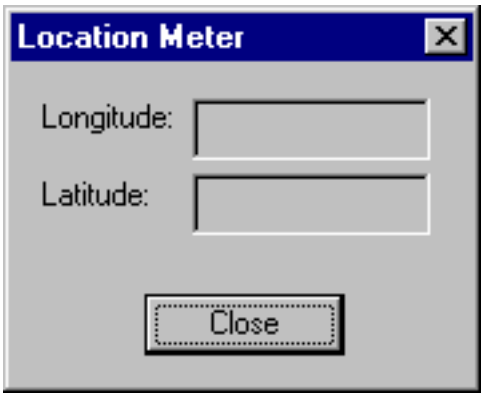
About the GPSNK Application

The GPSNK application was created using the MFC AppWizard specifying a dialog-based application and WinSock Support. These illustrations show the settings selected in AppWizard Steps 1 and 2 to

create the application.



GPSNK contains a dialog box resource and class that is used to get vehicle location information.



Here is a table describing the dialog box controls and corresponding member variables that are used to get and transfer user information.

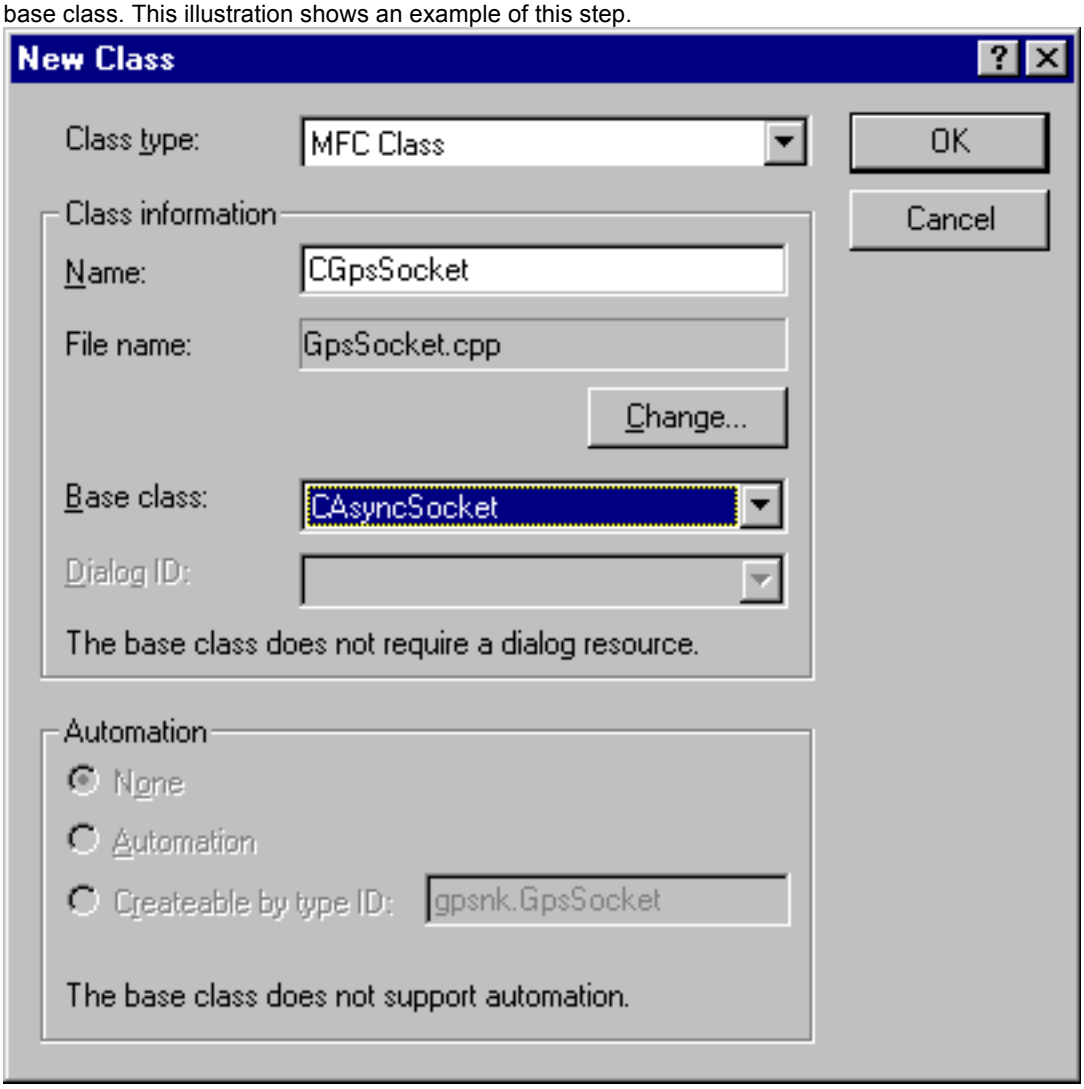
Location Information	Control Name	Member variable	Data Type
Longitude	IDC_EDIT_LON	m_strLongitude	CString
Latitude	IDC_EDIT_LAT	m_strLatitude	CString

There are two main steps required to complete the GPSNK application:

1. Create a socket class to receive the datagrams that the GPSRC application sends.
2. Add code to GPSNK's **InitInstance** function to construct, create, and initialize the socket.

➢ **Create a socket class to receive datagram packets**

1. Use ClassWizard to add a class to the application called **CGpsSocket**, using **CAsyncSocket** as the base class. This illustration shows an example of this step.



2. Add a **CDialog** pointer public member variable named m_pDlg to the **CGpsSocket** class.
   a. In the ClassView pane, right-click **CGpsSocket**, select Add Member Variable.
   b. In the Variable Type box, type **CDialog***.
   c. In the Variable Declaration box, type **m_pDlg**.
3. Add this constructor to the **CGpsSocket** class:

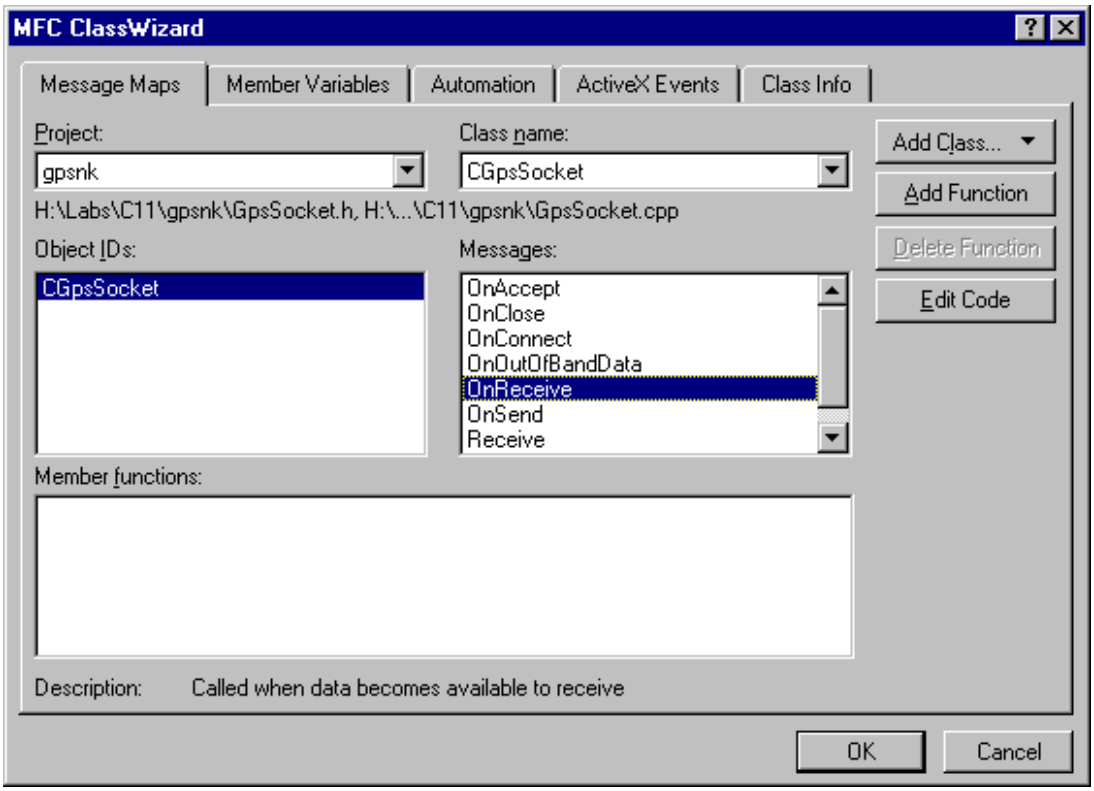
```
CGpsSocket::CGpsSocket(CDialog* pDlg)
{
    m_pDlg = pDlg;
}
```

4. Add a line to Gpsnk.Cpp to include the header file for the **CGpsSocket** class:

```
#include "GpsSocket.H"
```

5. Add a member function to the **CGpsSocket** class to handle the OnReceive message. The **OnReceive** function will be called when a datagram arrives at a socket.
   a. Press CTL+W to invoke the MFC ClassWizard dialog box and then click the Message Maps tab.
   b. Make sure that **CGpsSocket** is selected in the Class name box.
   c. From the Messages list, select OnReceive, click Add Function, and then to prepare for Step 3, click Edit Function.

      This illustration shows an example of this step.



6. To the body of the **OnReceive** function, *before* the call to the base class's **OnReceive** function, add code to extract the data and update the dialog box values.

```
// Buffer to hold incoming packet.
char buf[512+1];
int num= Receive(buf, 512);
buf[num] = '\0';// terminate data string with a NULL.

// Expected format of data: "DDD1:MM1:SS1 DDD2:MM2:SS2"
// where:
//  DDD1:MM1:SS1 is the degree/minute/second longitude value
//  DDD2:MM2:SS2 is the degree/minute/second latitude value
CString strLon;
CString strLat;

// Extract each coordinate as a separate string.
char* sep= " ";     // the string separator is a single space
char* p= strtok(buf,sep);    // first string
strLon= p;
p= strtok(NULL, sep);        // second string
strLat= p;
```

```
// Update the edit controls.
CEdit* pEdit;
pEdit= (CEdit*) m_pDlg->GetDlgItem(IDC_EDIT_LON);
pEdit->SetWindowText(strLon);
pEdit= (CEdit*) m_pDlg->GetDlgItem(IDC_EDIT_LAT);
pEdit->SetWindowText(strLat);
```

➢ **Construct, create, and initialize the socket**

1. Use the Developer Studio editor to open the file Gpsnk.Cpp.
2. In the **CGpsnkApp::InitInstance** function supplied by AppWizard, after the dialog construction (`m_pMainWnd = &dlg;`), but before the call to the **DoModal** function (`int nResponse = dlg.DoModal();`), add code to construct a socket, passing it the address of the dialog object:

```
// Construct the socket passing in a pointer to the dialog.
// The socket class will use this pointer to update the dialog.
CGpsSocket* pSocket= new CGpsSocket(&dlg);
```

3. Following the socket construction code, add the code to create a datagram socket with a preselected port number of 1088.

```
// Create the socket specifying a pre-selected port number and datagram-
type socket.
pSocket->Create(1088, SOCK_DGRAM);
```

4. Next, add code to initialize the socket's listening queue.

```
// Now setup listening queue for incoming datagram.
pSocket->Listen();
```

5. After the **DoModal** function call, add code to delete the socket before the application exits.

```
...
int nResponse = dlg.DoModal();
delete pSocket;
```

➢ **Build and test your application**

You can now build and test the GPSNK application. The Location Meter dialog box should appear with the string "waiting..." displayed in the two edit controls. At this point, the application is listening to its port 1088. No positional information will appear in the dialog box until you create the GPSRC application in Exercise 2 and run it in conjunction with GPSNK. The completed code for this exercise is in \Labs\C11\Lab04\Ex01.

Exercise 2: Creating a Source Socket Application

The purpose of this lab is to use the MFC WinSock classes to create two applications that implement a transmitter/receiver socket pair to simulate a vehicle location system based on the Global Positioning System.

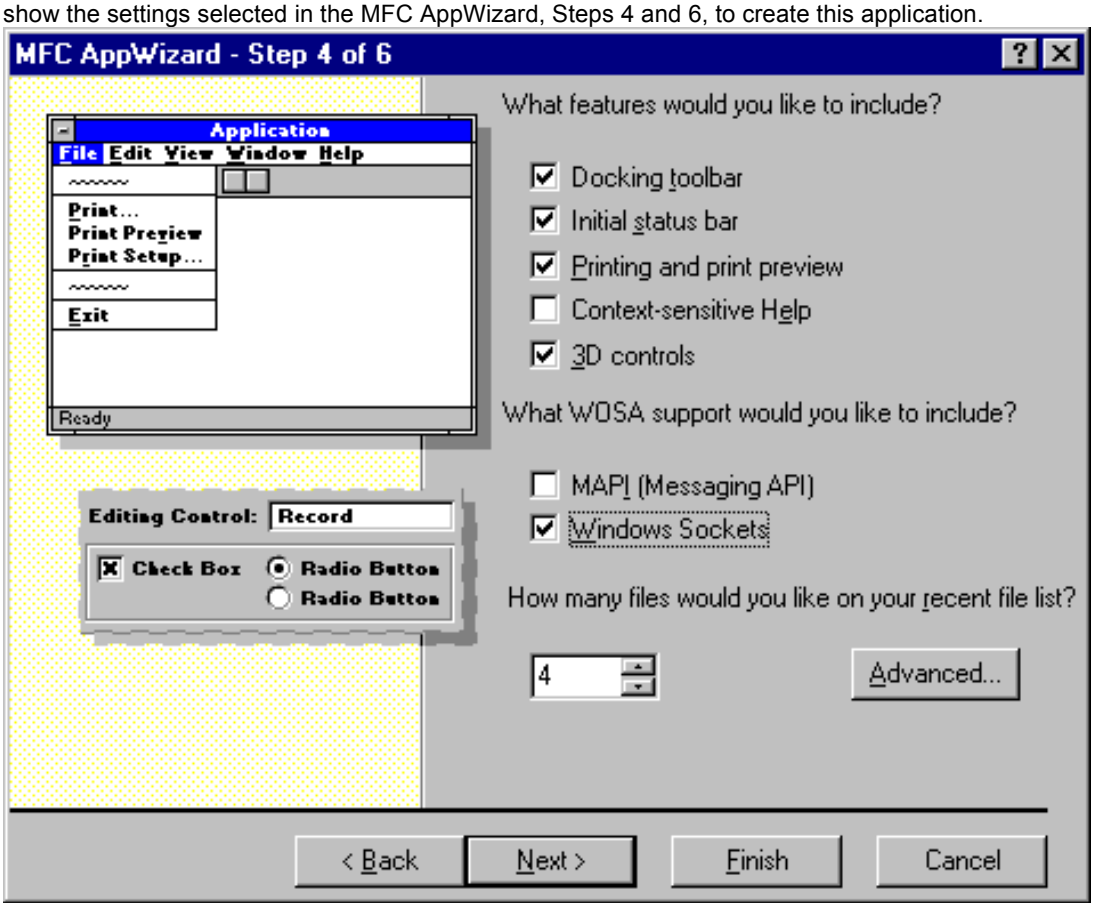
You will simulate a satellite/vehicle/wireless-link subsystem with a datagram "source" socket, that is, a socket that only sends packets. The base station will be simulated by a datagram "sink" socket, one that only receives packets.

In this exercise, you will complete the baseline application GPSRC, which simulates the satellite/vehicle/wireless-link subsystem. In GPSRC, you will create a datagram "source" socket, that is, a socket that only sends packets. It transmits vehicle location data to the GPSNK base station application that you create in Exercise 1.

About the GPSRC Baseline Application

For an overview of the lab scenario and the GPSNK application, refer to the instructions for Exercise 1 of this lab. In this exercise, you will modify the baseline application GPSRC.

The GPSRC application was created using the MFC AppWizard specifying a single document interface (SDI) application, WinSock support, and a view derived from the **CScrollView** class. These illustrations show the settings selected in the MFC AppWizard, Steps 4 and 6, to create this application.



This table describes member variables that have been added to the GPSRC baseline application's document class. The variables are used to embed a source socket object and the vehicle's positional data into the **GpsrcDoc** class.

Member Variable Name	Data Type	Description
m_pSock	CAsyncSocket*	This variable holds a pointer to the source socket object. Since the GPSRC application does not have to add or override functions of the **CAsyncSocket** class, it can use the class directly.
m_LonLat	CStringArray	This array holds the positional data that simulates the information that would be broadcast by a satellite/vehicle/wireless-link subsystem for a company's vehicle.
m_curLonLat	int	This integer is used as an index into the array m_LonLat.
m_strings	CStringArray	This variable holds the strings

	that will be displayed in the GPSRC application's view.

This table describes functions provided with the GPSRC baseline application that you modify or that provide application functionality related to the satellite/vehicle/wireless-link subsystem simulation.

Function Name	Description
CGpsrcDoc::OnNewDocument	You add code to this function to construct and initialize a datagram-type socket.
CGpsrcDoc::SendPosition	You add code to this function to send data out the socket. This function is called by a timer object.
TimerProc	You add code to this function to call CGpsrcDoc::SendPosition. This is a timer callback function.
CGpsrcView::OnCreate	You add code to this function to initialize a timer object.
CGpsrcView::OnDraw	You do not need to modify this function. The function is provided in its entirety. It displays the data being transmitted via the socket to the application view. This enables you to compare the data being sent by the GPSRC application with the data being received by the GPSNK application.

There are three main steps required to complete the GPSRC application:

1. Add code to construct and initialize the socket.
2. Add code to send data out the socket.
3. Add code to send the packet using a timer callback procedure.

➢ **Construct and initialize the socket**

Add code to the document class's **OnNewDocument** member function to create and initialize the socket.

1. Use the Developer Studio editor to open GpsrcDoc.Cpp. Find the function **OnNewDocument**.
2. After the statement `m_curLonLat = 0;` add code to construct a **CAsyncSocket** object, and code to initialize the socket using 0 for the port number and a datagram type of socket.

```
// Initialize GPS Socket as a datagram-type socket.
// Specifying port 0 will cause the Create function to assign
// a port number to this socket.
m_pSock= new CAsyncSocket;
m_pSock->Create(0, SOCK_DGRAM);
```

➢ **Send positional data out the socket**

To the bottom of the **if** block in the document class member function **SendPosition**, add code to send the datagram. To maintain simplicity, you can assume that the receiving application (GPSNK) resides on the same computer as the GPSRC application.

```
// Send out datagram assuming the receiving socket (server)
// resides on the same computer as this application does.
char server[512];
```

```
gethostname(server,511);
m_pSock->SendTo(lonlat, lonlat.GetLength(), 1088, server);
```

➢ **Send the packet**

1. Add code to the empty timer callback procedure, **TimerProc**, to call the document class function **SendPosition** to send a packet. **TimerProc** is defined in the view class's implementation file, GpsrcView.Cpp.

```
// Get the document object
CGpsrcDoc* pDoc=
(CGpsrcDoc*) ((CMainFrame*) AfxGetMainWnd())->GetActiveDocument();
// Tell document to send out a packet
pDoc->SendPosition();
```

2. To complete the code added in Step 1, add an include directive for the **CMainFrame** class's header file to the top of GpsrcView.Cpp.

```
#include "mainfrm.h"
```

3. In the view class's **OnCreate** function, after the call to the base class's **OnCreate** function, add code to initialize the timer with an ID of 1234, an interval of 1,000 milliseconds, and to use the callback procedure referred to in Step 1.

```
// Add timer initialization for 1000 milliseconds
// and specifying our timer procedure to handle the event.
// The timer identifier, 1234, is not used.

SetTimer(1234, 1000, &TimerProc);
```

➢ **Build and test your application**

You are now ready to build your application and test it by running it in conjunction with the GPSNK application created in Exercise 1.

Once you have built the GPSRC application, you can test it, by first running GPSNK. Once the GPSNK Location Meter dialog box is displayed, move it to a corner of your screen and then run the GPSRC application. You should see that the positional data in view of GPSRC is transmitted to GPSNK and displayed in the Location Meter dialog box. The completed code for this exercise is in \Labs\C11\Lab04\Ex02.