ADENEO

HEADLESS BLUETOOTH MANAGER

SAMPLE APPLICATIONS USER MANUAL

Ref.: C000108 REV A

Document History

Revision	Date	Author	Version Follow-up
A	Jan 21, 2008	RMarcilla	Initial version

Table of Contents

Adeneo Headless Bluetooth Manager

1. Sample Applications User Manual

1.1 Setting up a CE6 OS Design for use with Bluetooth

Be sure to start with an OS Design that already boots and runs correctly. Change to the catalog components view, and add the following items:

Under “Core OS – CEBASE – Communication Services and Networking – Personal Area Network (PAN) – Bluetooth – Bluetooth Protocol Stack with Transport Driver Support” add “Bluetooth Stack with Integrated CSR Chipset Driver” and either “Bluetooth Stack with Integrated USB Driver” or “Bluetooth Stack with Integrated UART Driver” depending on whether the Bluetooth adapter is connected via USB or serial

Under “Core OS – CEBASE – Applications and Services Development – C Libraries and Runtimes” add “C++ Runtime Support for Exception Handling and Runtime Type Information”

Under “Core OS – CEBASE – Component services – Component Object Model” add “COM” and “DCOM”

Under “Core OS – CEBASE – XML – MSXML 3.0” add “XML Core Services and Document Object Model (DOM)”

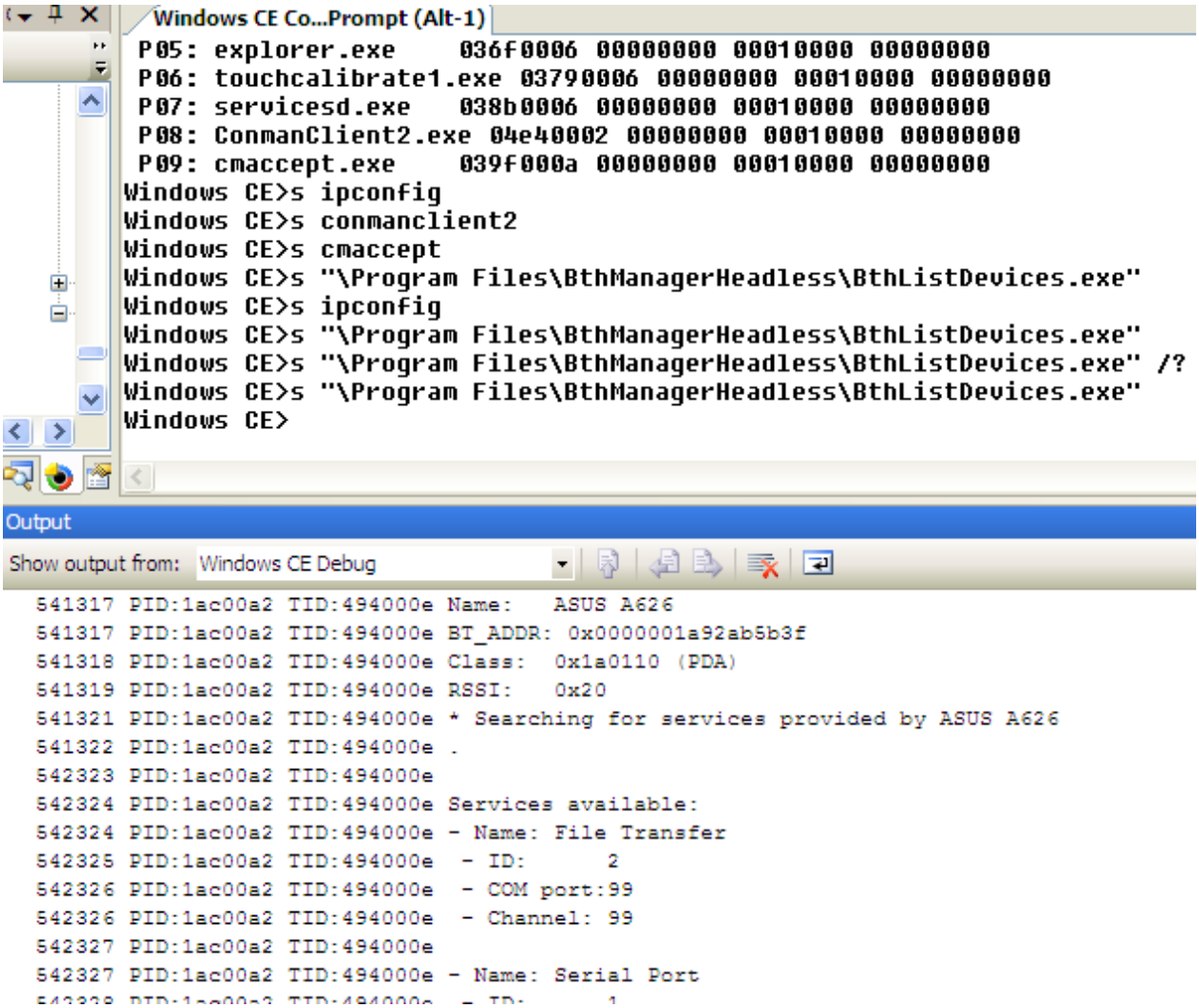
Under “Core OS – CEBASE – Communication Services and Networking – Personal Area Network (PAN) – Bluetooth – Bluetooth Profiles Support” add “HS/HF and Audio Gateway Service” for Bluetooth audio

Under “Core OS – CEBASE – Object Exchange Protocol (OBEX)” add “OBEX Client” and “OBEX Server” if OPP or FTP will be used. **Also add both items underneath OBEX Server**, or OBEX (OPP/FTP) servers will not function properly

1.2 Using the sample applications

Copy the sample executables and BthManagerDll.dll to the same directory on the device. They can be executed from the Command Window like any other windows program (see screenshot below).

Note that they are somewhat slow, because each time any of the sample applications is run it loads the Bluetooth stack, then has to find the device it communicates with all over again. (For example, if one runs BthListDevices.exe to find a name, then BthPush.exe with the name of the device , BthPush will have to search for devices all over again which will take time) With the GUI Bluetooth Manager or a custom Bluetooth enabled application, this would only be necessary once, so the interface will run much more quickly.



1.2.1 Sample 1: BthListDevices.exe: List Devices and Services

1.2.1.1 Summary

This is a relatively basic application which allows one to quickly verify that the Bluetooth driver is installed and working, as well as list available Bluetooth devices and services.

1.2.1.2 Usage

Just run the application without any parameters to list devices and for each device list all available services. By default, the application will not initiate or respond to pairing requests.

Using command line parameters, it's possible to toggle displaying information available on the local bluetooth adapter, attempt to pair with remote devices before listing their available services, or only list devices/services with a specified device name/bluetooth address.

1.2.1.3 Command Line Parameters

```
   By default, the application will display information on the local device, then list all remote devices and services
    /? -? /h -h: Display this help screen and exit
    /d -d: Print information on the local device
    /l -l: List remote devices
    /p -p ["pin"]: Attempt to pair with each remote device before listing services it has.  If PIN is not specified it will default to 42.  If a pin with spaces is entered, enclose it in double quotes.  PINs with double quotes and PINs that start with a slash ("/") or dash ("-") are not supported.
    /s -s: List remote services for each device (services will be listed for any device specified with rb or rn regardless of this option)


    Filtering options: (If rn or rb is specified, only devices that match the given name and/or address will be displayed)
    /rn "Name" -rn "Name": Search for a Bluetooth device called Name
    /rb [0x]BT_ADDR -rb [0x]BT_ADDR: Search for a Bluetooth device with a given 64 bit address specified in hex.  0x before the address is optional.  Don't include any spaces in the address
```

1.2.1.4 Developers' Notes

The important parts of this application are:

Create an instance of CDeviceLocal

```
      CDeviceLocal LocalDevice;
```

1.2.1.4.1 Activate the Bluetooth Adapter

```
      if (!LocalDevice.Activate(TRUE))
      {
            wprintf(L"Error activating the Bluetooth adapter\n");
            return 0;
      }
```

1.2.1.4.2 Initiate an Inquiry Thread for Remote Devices

```
  int InquiryCallback(LPARAM lParam, LPARAM lParam2)
  {
      gbInquiryComplete = TRUE;
      return 0;
  }
```

```
      // Register InquiryCallback with the DLL and WM_BTINQUIRY_OVER message
      //   so it will be called when the search is complete
```

```
	BTHMNG_RegisterCallback(WM_BTINQUIRY_OVER, InquiryCallback, NULL);

	// Begin a search for available remote devices.  When it is complete,
	//  InquiryCallback will be called and set gbInquiryComplete to TRUE
	gbInquiryComplete = FALSE;
	if (!LocalDevice.FindDevices())
	{
		wprintf(L"Error searching for devices\n");
	}
```

1.2.1.4.3 Wait for the thread to complete and send a WM_BTINQUIRY_OVER message (**Note**: this waiting could also be done with events, see BthPush.exe for an example of how to do it with CE events rather than polling global variables)

```
	while (!gbInquiryComplete) Sleep(500);
```

1.2.1.4.4 Looping through remote devices found

```
	ArrayDevices DeviceList = LocalDevice.GetDevices();

	for (ushort iDevice=0; iDevice<DeviceList.size(); iDevice++)
	{
		pDevice = DeviceList.at(iDevice);
```

1.2.1.4.5 Displaying information available about a device

(Note that BTAddr is a 64 bit variable. ClassToString() is defined in BthListDevices.cpp, it translates a class into "Phone", "PDA", etc)

```
	wprintf(L"Name:   %s\n", pDevice->GetName().c_str());
	wprintf(L"BT_ADDR:  0x%08x%08x\n", (DWORD) (pDevice->GetBtAddr()>>32), (DWORD) pDevice->GetBtAddr());
	wprintf(L"Class:    0x%x  (%s)\n", pDevice->GetClass(), ClassToString(pDevice->GetClass()));
	#if ( _WIN32_WCE >= 0x600)
		wprintf(L"RSSI:   0x%x\n", pDevice->GetRSSISignal());
	#endif
```

1.2.1.4.6 Pairing with a Remote Device

```
	if (pDevice->IsPaired())
		wprintf(L"Already paired with %s\n", pDevice->GetName().c_str());
	else
	{
		pDevice->AskPairRequest(iPinLength, sPin);

		if (pDevice->IsPaired())
		{
			wprintf(L"Paired successfully\n");
		}
		else
		{
			wprintf(L"Unable to pair with device\n");
		}
	}
```

1.2.1.4.7 Initiate an Inquiry Thread for Remote Services

```
gbInquiryComplete = FALSE;
if (!pDevice->FindServices())
{
      wprintf(L"Error, could not start search thread\n");
      continue;
}
```

1.2.1.4.8 Displaying information available about a service

```
pService = ServiceList.at(iService);
wprintf(L"- Name: %s\n", pService->GetName().c_str());
wprintf(L" - ID:      %d\n", pService->GetID());
wprintf(L" - COM port:%d\n", pService->GetComPort());
wprintf(L" - Channel: %d\n", pService->GetChannel());
wprintf(L"\n");
```

1.2.1.4.9 Wait for the thread to complete (This could also be done with an event ; in this application it's primarily done with the while/sleep to allow for printing periods to indicate progress)

```
while (!gbInquiryComplete) Sleep(500);
```

1.2.2 Sample 2: BthConnectSpp.exe: Find and connect to a Serial Port Profile

1.2.2.1 Summary

T his application finds and connects to a remote Serial Port Profile server.

1.2.2.2 Usage

By default, it requires a device name or bluetooth address to be specified with /rn or /rb (respectively). It connects to the device without pairing, creates a virtual COM port, and sends “Hey there\r\n” every second for two minutes.

The Bluetooth address is a 64 bit value which is written in hex and the format for entering it is identical to the format in which it’s displayed by BthListDevices.exe. When entering a name with spaces, be sure to enclose the entire name in double quotes. Names and addresses are case insensitive.

1.2.2.3 Command Line Parameters

```
   By default, the application will display information on the local device, then search for a remote device and attempt to connect to its SPP server.  Once the SPP connection is established, the application opens the COM port and sends "Hey there\r\n" every second for two minutes
    /? -? /h -h: Display this help screen and exit
    /m -m: Don't send a default message or open the COM port created, just leave it open for other applications
    /t -t number: Time to leave the connection open in seconds
    /c -c port: COM port to use
    /p -p ["pin"]: Attempt to pair with the remote device before listing services/attempting to connect via SPP.  If PIN is not specified it will default to 42.  If a pin with spaces is entered, enclose it in double quotes.  PINs with double quotes and PINs


    Filtering options: (If rn or rb is specified, the program will attempt to open a serial connection only with devices that have the specified address/name)
    /rn "Name" -rn "Name": Case insensitive search for a Bluetooth device called Name
    /rb [0x]BT_ADDR -rb [0x]BT_ADDR: Search for a Bluetooth device with a given 64 bit address specified in hex.  0x before the address is optional.  Don't include any spaces in the address
```

1.2.2.4 Developers’ Notes

The important parts of this application are:

1.2.2.4.1 Callback for answering Pair requests

```
//-------------------------------------------------------------------
//! \brief Callback for handling PIN requests
//!
//! \param lpDevD a pointer to the remote device the query is coming from
//! \param lpDevD2 unused, pointer passed in when the callback was registered
//!
//! \return 0
//-------------------------------------------------------------------
int OnDevicePairRequest(LPARAM lpDevD, LPARAM lpDevD2)
{
    CDeviceDistant *pDeviceDistant = NULL;

    wprintf(L"\nAnswering pairing request with pin: \"%s\"\n", sPin);

    if (lpDevD != NULL)
    {
          pDeviceDistant = (CDeviceDistant*) lpDevD;
```

```
            pDeviceDistant->AnswerPairRequest(iPinLength, sPin);
      }

      return 0;
  }
```

1.2.2.4.2 Obtaining a list of available COM ports

```
      vectorPortCom = pSpp->getAvailablePortComList();

      if (vectorPortCom.size() < 1)
      {
            wprintf(L"Error, no COM ports available");
      }
      else
      {
            wprintf(L"COM ports available: %d", vectorPortCom[0]);
            for (ushort iCOMiterator = 1;
                        iCOMiterator < vectorPortCom.size(); iCOMiterator++)
                  wprintf(L", %d", vectorPortCom[iCOMiterator]);
            wprintf(L"\n");
      }
```

1.2.2.4.3 Connecting to a remote SPP server

```
      pSpp->Connect();
```

1.2.3 Sample 3: BthServices.exe: Host SPP, FTP, and OBEX Push Profiles

1.2.3.1 Summary

This application allows one to start and stop SPP, OPP, and FTP servers

1.2.3.2 Usage

By default, just running the application will enable the SPP and OPP services with authentication not required and will respond with a PIN of 42 when remote devices attempt to pair with the local device.

If any parameters are specified, no services will be started by default. (so “BthServices.exe” starts SPP and OPP, but “BthServices.exe /a /p 1234” will not start any services until instructed to

The application continues to run in the background until instructed to stop by running “BthServices.exe /stop”, even if all running services are stopped

SPP, OPP, and FTP services can be started and stopped while BthServices is running by running a new instance of the application with the correct flags. When the application runs, if it detects that it’s not the first instance it will just set system-wide events to communicate with the original instance of the application rather.

To change the PIN or Authentication, it is necessary to stop the application completely with “BthServices.exe /stop” and then start it again with the proper /p or /a flags

1.2.3.3 Command Line Parameters

```
   By default, the application will start SPP and OPP services and run until instructed to stop by another
instance of the application.  If any services are specified only those services specified will start.
    (For all command line parameters, "-" can be substituted for "/".)
     /? /h: Display this help screen and exit.  (This screen will also be displayed if there are any
errors parsing the command line)
     /stop: Stop all services and running instances of the application and exit
     /(service) stop: If "stop" is specified, the service will be stopped if it's running
     /spp [stop]: Serial Port Profile
```

```
    /sppc Number: COM port to host SPP over (also starts /spp if /spp is not specified, but ftp and opp will not be started unless directed to).  Note: If /sppc is executed in a second instance of the application, the SPP server will be restarted with the pr
    /ftp [stop]: File Transfer Profile
    /opp [stop]: Object Push Profile
    /p "pin": Pin to use if one is requested
    /a: Require PIN authentication.  If a pin is not specified with /p it will default to 42
```

1.2.3.4 Developers' Notes

The important parts of this application are:

1.2.3.4.1 Start a service

```
if(gpServerFTP->IsConnected())
{
      wprintf(L"FTP Server already started, stopping it..\n");
      gpServerFTP->Disconnect();
}

gpServerFTP->SetAuthenticated(gbAuthenticate);

gpServerFTP->Connect();
```

1.2.3.4.2 Stop a service

```
if(gpServerFTP->IsConnected())
{
      wprintf(L"Stopping FTP server\n");
      gpServerFTP->Disconnect();
}
else
      wprintf(L"FTP server already stopped\n");
```

1.2.4 Sample 4: BthPush.exe: Push a file to a specified Bluetooth device

1.2.4.1 Summary

This application connects to a specified Bluetooth device and sends a file via Object Push Profile (OPP)

1.2.4.2 Usage

First use BthListDevices.exe to obtain the name or Bluetooth address of a device, then run the application with the name or address along with the name of the file to find the remote device and start the upload. After the upload has finished, the application will exit.

1.2.4.3 Command Line Parameters

```
  Usage: BthPush.exe [/p /rn /rb] "filename"
    Connects to a remote device and attempts to send a file specified by filename
    /? -? /h -h: Display this help screen and exit
    /p -p ["pin"]: Attempt to pair with the remote device before listing services/attempting to connect. If PIN is not specified it will default to 42.
   If a pin with spaces is entered, enclose it in double quotes.  PINs with double quotes and PINs that start with a slash ("/") or dash ("-") are not supported.

    Filtering options: (One of either rn or rb is required)
    /rn "Name" -rn "Name": Case insensitive search for a Bluetooth device called Name
    /rb [0x]BT_ADDR -rb [0x]BT_ADDR: Search for a Bluetooth device with a given 64 bit address specified in hex.  0x before the address is optional.  Don't include any spaces in the address
```

1.2.4.4 Developers' Notes

The important parts of this application are:

1.2.4.4.1 Callback function for viewing file transfer status

```
  //---------------------------------------------------------------------
  //! \brief Callback for seeing the status of a file transfer
  //!
  //! \param iPercentTransferred   Percentage of the file transferred
  //! \param dRate                       kilobytes/second
  //! \param lParam                      unused
  //!
  //! \return 0
  //---------------------------------------------------------------------
  static void TransferCallback(int iPercentTransferred, double dRate, LPARAM lParam)
  {
      if (iPercentTransferred == BTH_TRANSFER_STARTED)
      {
            iPercentTransferred = 0;
            if (ghTransferStarted != NULL)
            {
                  SetEvent(ghTransferStarted);
            }
            wprintf(L"\n --- Transfer started ---\n", dRate);
      }
      else if (iPercentTransferred == BTH_TRANSFER_COMPLETE)
      {
            iPercentTransferred = 100;
            if (ghTransferComplete != NULL)
            {
                  SetEvent(ghTransferComplete);
            }
            wprintf(L" --- Transfer complete  ---\n");
      }
      else if (iPercentTransferred == BTH_TRANSFER_SIZEUNKNOWN)
      {
            //Just print dots so that it's clear the transfer is progressing, because
there's no
            // information available on the rate/size
            wprintf(L".");
      }
      else
            wprintf(L"Transferred:  %d%%,  Rate:  %01.2fk/s\n",  iPercentTransferred,
dRate);
  }
```

1.2.4.4.2 Initiating an object push

```
  pOpp->PushObject(sFileName, TransferCallback, NULL)
```

1.2.5 Sample 5: BthFTP.exe: File Transfer Profile Client

1.2.5.1 Summary

This application is a basic command line File Transfer Profile client. It supports uploading, downloading, and listing the contents of remote directories.

1.2.5.2 Usage

First find the name or Bluetooth address of a device that supports FTP by using BthListDevices.exe, then run BthFtp using the specified name or address.

Each time the program is run, it will search for available remote devices, initiate a new connection, execute the action specified on the command line, and then exit. This means that each operation is completely independent of any other, but it also means that each operation takes roughly 10-30 seconds. It would be straightforward to redesign the application using a GUI or terminal interface to keep a connection open and execute multiple commands without restarting the connection, in which case operations would be executed much more quickly, as in the MFC-based Bluetooth Manager GUI.

1.2.5.3 Command Line Parameters

```
   Usage: (Note: parameters are not case sensitive)
    If /l, /s, and /g are unspecified, by default the program will list the files/folders in the root of the remote FTP server
     /? -? /h -h: Display this help screen and exit
     /p -p ["pin"]: Attempt to pair with the remote device using "pin".  If "pin" is not specified it will default to 42.  If a pin with spaces is entered, enclose it in double quotes.
     PINs with double quotes and PINs that start with a slash ("/") or dash ("-") are not supported.
     /l -l "Remote Path": List files/folders in remote path specified by "Remote Path".  For example: BthFTP.exe /p 339215 /rn "Home Computer" /f "\Images\Camera\October"
     /s -s "Local Path" "Remote Path": Send the file at "Local Path" on the local device to the remote device and store it in "Remote Path"
     /g -g "Remote Path" "Local Path": Get the file at "Remote Path" on the remote device and store it in "Local Path" (defaults to the current directory)


     Filtering options: (Either rn or rb is required to use FTP.)
     /rn "Name" -rn "Name": Case insensitive search for a Bluetooth device called Name
     /rb [0x]BT_ADDR -rb [0x]BT_ADDR: Search for a Bluetooth device with a given 64 bit address specified in hex.  0x before the address is optional.  Don't include any spaces in the address
```

1.2.5.4 Developers' Notes

The important parts of this application are:

1.2.5.4.1 Connecting to an FTP server

1.2.5.4.2 Listing the contents of a folder

```
  if (!pWorkingFolder->Browse())
  {
      wprintf(L"Error enumerating contents of folder");
      pFTP->Disconnect();
      continue;
  }

  wprintf(L"\n%d objects found:\n", pWorkingFolder->GetFtpObjects().size());

  CFtpFolder* pFolder = NULL;
  CFtpFile* pFile = NULL;
  CFtpObject* pFtpObject = NULL;
  for (unsigned int iFtpObject=0; iFtpObject<pWorkingFolder->GetFtpObjects().size(); iFtpObject++)
  {
      pFtpObject = pWorkingFolder->GetFtpObjects().at(iFtpObject);
      wprintf(L" Name: \"%s\", ", pFtpObject->GetName().c_str());
      switch (pFtpObject->GetID())
```

```
      {
      case OBEX_FTP_FILE:
            pFile = (CFtpFile*) pFtpObject;
            wprintf(L"(File), Size: %d bytes, Full Path: \"%s\\%s\"",
                  pFile->GetSize(),          pFile->GetPath().c_str(),          pFile-
>GetName().c_str());
            break;

      case OBEX_FTP_FOLDER:
            pFolder = (CFtpFolder*) pFtpObject;
            wprintf(L"(Folder), Full Path: \"%s\"", pFolder->GetPath().c_str());
            break;
      }
      wprintf(L"\n");
  }
```

1.2.5.4.3 Uploading a file

```
      pWorkingFolder = new CFtpFolder(pFTP, NULL, sRemotePath);
      pWorkingFolder->PutFile(sLocalPath, TransferCallback, NULL );
```

1.2.5.4.4 Downloading a file

```
      //Set the directory to download the file to
      pFTP->SetPath(sLocalPath);

      pFileToGet->Get(TransferCallback, NULL);
```

1.2.5.4.5 Deleting a file

Although it's not currently implemented, it would be easy to add deleting remote files to an FTP client. Just run a folder's DeleteObject() method on a CFtpObject pointer contained within a folder, like so:

```
      pWorkingFolder->DeleteObject(pFtpObject)
```

1.3 Building the sample applications

1. Install a CE6 SDK and modify the sln and vcproj files to use it

   This is an unfortunate requirement, as there is no Microsoft Standard SDK for CE6 development, It is necessary to create and install a CE6 SDK, then open the SLN and VCPROJ files in the HeadlessSamples folder with a text editor such as Notepad++ or PSPAD and replace all instances of “PXA270 DK SDK (ARMV4I)” with the name of the newly created SDK

2. **(This step should already be done for the binary version, but is necessary whenever a new version of BthManagerDll.dll is built)** Open the Property Pages for each project contained in the BthManagerHeadless solution and verify that for the current mode it’s being built in, the correct directory containing BthManagerDll.lib and .dll are referenced in “Linker -> General -> Additional Library Directories” and the correct header files are referenced in “C/C++ -> General -> Additional Include Directories”. It will also be necessary to have the DLL in the same directory as the sample application is deployed to, which is configured in “Configuration Properties ->Deployment” and “Configuration Properties -> Debugging”

3. Build the solution and deploy to the embedded device automatically using CoreCon/Activesync or by manually copying the files over. If using Visual Studio to debug, command line parameters are specified by right clicking the project name, then selecting “Debugging” under “Configuration Properties -> Debugging”