Remote Data Acquisition with a PDA

The Development of a Simple System for Data Collection Over a Network

Michael Lekon
Instructor: Janusz Zalewski
Fort Myers, Florida
12/4/2009
# Table of Contents

Introduction .................................................................................................................................................. 3

Overview ................................................................................................................................................... 3

Terminology .............................................................................................................................................. 6

Problem Description ..................................................................................................................................... 7

Solution ......................................................................................................................................................... 9

PDA Data Acquisition ................................................................................................................................ 9

Preparations for Development .................................................................................................................. 10

Data Acquisition Program .......................................................................................................................... 13

Adding Network Capabilities .................................................................................................................... 17

The Client Program .................................................................................................................................... 20

The Server Program .................................................................................................................................... 23

Experimentation ......................................................................................................................................... 27

Conclusion ................................................................................................................................................... 29

References .................................................................................................................................................. 30
Introduction

Overview

In recent years, handheld computing devices have exploded in popularity. Despite their limitations, developers are still able to produce robust applications that take advantage of the portable nature of the devices. One such utilization of this technology is to provide a mobile platform for acquiring data. This allows reading to be taken “in the field” and analyzed in real time. Combine this with a network connection, and the data can be instantly transmitted to the lab for more permanent store and further analysis. This document focuses on implementing a system capable of acquiring data with a mobile device and sending it over a network to a designated server program running on a PC. An overview of the system layout is shown in Figure 1.

![Diagram](image_url)

Figure 1: Data is acquired from the DAQ card, processed by the PDA, transmitted to the nearest access point, and forwarded to the PC running the program to receive these data.
For the purposes of this demonstration, a single set of hardware is used.

- **Mobile Device** - Dell Axim X50 with the Windows Mobile 2003 operating system. Seen in Figure 1.
- **Data Acquisition Device** - NI CF-6004 Compact Flash card. Seen in Figure 2
- **Connector Block** - Phoenix Contact Subcon 15 connector block. Seen in Figure 2.
- **Development and Server System** – Hewlett Packard Desktop PC running the Windows XP operating system
- **Voltage Source** – 4x AAA Batteries.

![Image of Dell Axim X50 PDA, Compact Flash Data Acquisition Card, and Phoenix Contact Subcon 15 connector block.](image)

**Figure 2**: Dell Axim X50 PDA (left) Compact Flash Data Acquisition Card (center) and Phoenix Contact Subcon 15 connector (right)

The technical specifications of the Axim X50 are presented in Table 1, following manufacturer specifications:

<table>
<thead>
<tr>
<th>Installed RAM</th>
<th>64MB SDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installed ROM</td>
<td>128MB</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel 520 MHz XScale</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Input device type</td>
<td>Stylus, Touch screen, 5-way navigation button</td>
</tr>
<tr>
<td>Digital audio standards supported</td>
<td>MP3, WMA</td>
</tr>
<tr>
<td>Audio input type</td>
<td>Built in Microphone</td>
</tr>
<tr>
<td>Audio output type</td>
<td>Built in Speakers</td>
</tr>
<tr>
<td>Voice recording capability</td>
<td>Yes</td>
</tr>
<tr>
<td>Display type</td>
<td>3.5 in TFT active matrix</td>
</tr>
<tr>
<td>Color support</td>
<td>16-bit(64K colors)</td>
</tr>
<tr>
<td>Max resolution</td>
<td>240x320</td>
</tr>
<tr>
<td>Wireless connectivity</td>
<td>Wi-Fi, Bluetooth, IEEE 802.11b</td>
</tr>
<tr>
<td>Battery Type</td>
<td>1100mAh Lithium Ion</td>
</tr>
<tr>
<td>Storage Card</td>
<td>SD Memory Card, Compact Flash Card Type II</td>
</tr>
</tbody>
</table>

The Axim X50 features a 3.5" QVGA display with a 320x240 resolution, and includes the power and storage capacity sufficient for our purposes. The X50 handheld has CF (Compact Flash) and SDIO expansion slots for connecting scanners, GPS systems, network cards, modems, and other external devices. The X50 is powered by the Intel PXA270 processor and is pre-installed with Microsoft Windows Mobile 2003 Second Edition operating system.

NI CF-6004 CompactFlash Data Acquisition for PDAs is a small 16cm^2 card which features 4 analog input channels with 14-bit resolution and up to 132 kilosamples per second aggregate sampling rate. A general view of the card’s inner workings is shown in Figure 3, and a description of its pins is shown in Figure 4.
Figure 3: General view of the CF-6004 DAQ board

Figure 4: Pin details for the CF-6004

<table>
<thead>
<tr>
<th>Module</th>
<th>Terminal</th>
<th>Signal</th>
<th>Wire Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AI GND</td>
<td></td>
<td>White or Red</td>
</tr>
<tr>
<td>2</td>
<td>AI 0</td>
<td></td>
<td>White/Black</td>
</tr>
<tr>
<td>3</td>
<td>AI 1</td>
<td></td>
<td>Red/Black</td>
</tr>
<tr>
<td>4</td>
<td>AI 2</td>
<td></td>
<td>Yellow/Black</td>
</tr>
<tr>
<td>5</td>
<td>AI 3</td>
<td></td>
<td>Green/Black</td>
</tr>
<tr>
<td>6</td>
<td>RSRVD</td>
<td></td>
<td>Green or Blue</td>
</tr>
<tr>
<td>7</td>
<td>RSRVD</td>
<td></td>
<td>Blue/Black</td>
</tr>
<tr>
<td>8</td>
<td>+3.3 V</td>
<td></td>
<td>Brown/White</td>
</tr>
<tr>
<td>9</td>
<td>NC</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>D GND</td>
<td></td>
<td>Orange or Gray</td>
</tr>
<tr>
<td>11</td>
<td>PFI 0</td>
<td></td>
<td>Orange/Black</td>
</tr>
<tr>
<td>12</td>
<td>PFI 1</td>
<td></td>
<td>Gray/Black</td>
</tr>
<tr>
<td>13</td>
<td>PFI 2</td>
<td></td>
<td>Purple/White</td>
</tr>
<tr>
<td>14</td>
<td>PFI 3</td>
<td></td>
<td>Pink/Black</td>
</tr>
<tr>
<td>15</td>
<td>D GND</td>
<td></td>
<td>Purple or Pink</td>
</tr>
</tbody>
</table>

Terminology
Problem Description

As stated above, the goal is to create a user-friendly system for collecting data with a mobile device and transmitting those data to a more permanent location.

In order to accomplish this several pieces of hardware and software are required. Hardware consists of:

- A mobile device to receive process and transmit data, while providing the ability to present the user with a graphical user interface.
- A method to translate external signals into digital data which can be interpreted by the mobile device.
• A device to accept data from the mobile device which has access to a long-term storage system.
• Wire and/or wireless network connections linking the two devices.

In developing the software, the following requirements must be met:
• A networking protocol shall be utilized such that data arrive reliably and is supported by both devices
• A software platform shall be selected which facilitates easy and efficient development.
• On the mobile device:
  o The user shall have the ability to start and stop data collection.
  o The user shall be able to specify the address of the data’s destination before collection begins.
  o Data collected shall be displayed for the user in real time.
  o Data shall be transmitted within 5 seconds of being collected.
  o A minimum amount of processing shall be done due to the power concerns of mobile devices
• On the immobile device:
  o A pre-defined port shall be listened on for incoming connections from the mobile device.
  o When a connection is established, data shall be read until the connection is closed or no new data arrives within 5 seconds.
  o Data which has been received shall be displayed for an operator to monitor the system’s status.
Solution

Two computing devices are directly involved in this system (the PDA client and the PC server), so software for both must be developed.

The PDA software must acquire data from the CF-6004 card, and send the data through a network connection. The PC software must continually listen for and read incoming data from the PDA.

PDA Data Acquisition

Communication with the CF-6004 device is done through an API provided by the DAQmx Base package. These functions operate on “tasks”, which are configurations of one or more channels on the device. Tasks are described in greater detail later. For this simple demonstration, only three of these functions are used: Start Task, Stop Task, and Read. A brief description of each is given in Table 2.

<table>
<thead>
<tr>
<th>Function Name and Description</th>
<th>Connector Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAQmx Base Read</td>
<td>task/channels in is the name of the task to which the operation applies. timeout specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the VI returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. task out is a reference to the task after this VI or function runs. data returns a 1D array of samples. Each element of the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.</td>
</tr>
<tr>
<td>Analog 1D DBL NChan 1Samp</td>
<td>task/channels in is the name of the task to which the operation applies. timeout specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the VI returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. task out is a reference to the task after this VI or function runs. data returns a 1D array of samples. Each element of the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.</td>
</tr>
</tbody>
</table>

Table 2: DAQmx Base Functions
Transitions the task to the running state to begin the measurement or generation. Using this VI is required for all applications in NI-DAQmx Base.

**DAQmx Base Stop Task**

Stops the task and returns it to the state the task was in before the DAQmxBase Start Task VI ran. Using this VI is required for all the applications in NI-DAQmx Base.

In addition to these VIs, the API also provides a DAQmx Base Task constant. This constant allows a task created in Task Configuration Utility to be specified and started. Though this can be changed to a control, a constant is sufficient for a simple demonstration.

**Preparations for Development**

Unlike in previous chapters, this time, due to the limitations of the PDA, we can’t simply install the LabVIEW IDE (integrated development environment) and develop from there. Instead, development must be done on a host computer, such as a typical desktop or laptop, using a software development kit (SDK) which allows the program to execute on the target platform, that is, a PDA. A mobile device development toolkit module for LabVIEW is available from National Instruments, and provides the ability to develop for a wide variety of mobile devices. In addition a set of LabVIEW drivers need to be installed on the target device.
In addition to that, data acquisition using the National Instruments CF-6004 DAQ Card requires several software packages to be installed on the development host. Since we use a Windows-based host, that’s what is described here. Other than LabVIEW and the LabVIEW Mobile module, the DAQmx Base device drivers must be installed on the PC host as well as the mobile device. The versions of the DAQmx Base software have a narrow window of compatibility with versions of LabVIEW. LabVIEW 8.2.1 requires DAQmx Base 2.2, for example. Once this is done, the Microsoft ActiveSync software (or Windows Mobile Device center, if using Windows Vista) must be installed. The device drivers must now be installed on the PDA. This is done using an installation utility provided by the DAQmx Base software when a link between the PC and PDA has been established. The utility is typically found in the Windows Start Menu under “Programs – National Instruments – NI-DAQmx Base – Utilities – Windows Mobile Driver Installation”.

Once the software is set up, several additional steps must be taken during development. First, a new “Mobile Project” must be created and set up for the type of PDA being used (Figure 5). The Dell Axim X50 has a “Mobile Portrait Screen”, that is, it is taller than it is wide, and it is a “Pocket PC 2003 Device”. The location and names of the project file and the initial mobile VI can also be specified if desired. Finally, the option to create a build specification is given, which will automatically prepare the program to be built into an executable file. When this is done, in order to use the DAQmx Base API, tasks must be created using the DAQmx Base Task Configuration Utility (Figure 6), which is accessible from the Tools menu of a LabVIEW Mobile project. A task in this context is a pre-defined configuration of an input channel (of which the DAQ card has several). New tasks may be created for specific channels and attributes. To use
these tasks, a task constant or control must be created and wired to DAQmx Base VIs. The project can now be built and the resulting executable copied to the PDA’s memory.

The tasks used in this section are very simple. In the Task Configuration Utility, click Create New Task, name the task “ai1”, select the device (NI 6004), set the acquisition type to “analog input”, and click OK. Of the two tabs below the device name, only the AI Configuration is of use to us. Check only the Channel Enable “0” box, set the Scan Rate to 200 Hz, and change the Number of Scans to 1. Repeat this two more times, each time incrementing the name and Channel Enable selection (the second task can be named “ai2” with only channel 1 enabled).

Figure 5: The wizard for setting up a new Mobile Project.
Figure 6: The task configuration utility.

Data Acquisition Program

Figure 7: Code for PDA DAQ using DAQmx Base
This is a simple single-VI program. If not already open, double click the VI shown in the project outline. Due to the dimensions of the PDA’s screen, the front panel of a mobile VI is set to a minimum size. In addition, the size of most controls and indicators is increased for better use with the target’s small screen (Figure 8). This program reads from the task specified in a constant when the user clicks the “Run” button displaying the data on the waveform chart, stops reading with the “Stop” button, and exits the program with the “Exit” button.

The block diagram may already contain a small amount of code; delete this and place your own While loop in its place. The “Exit” button is simply wired to the Loop Condition, and the “Run”
button is wired to the case selector of a case structure within the loop. A 100 millisecond wait prevents the program from using all of the PDA’s limited processing power.

When the “Start” button is pressed, the True case executes, which is where the bulk of the program is placed. First, a task must be indicated and started. Place the Start Task VI and a Task Constant. In the Task Constant, type the name of a task that has been previously configured, such as “ai1” or “ai2”. Once the task has been started, the data acquisition loop begins. On each iteration, a call to the read function is done. Because DAQmxBase Read is a polymorphic function, the version which we want to execute must be decided, in this case, during development. Click the downward pointing arrow in the text area just below the VI’s icon. This will display a menu from which all of the possibilities can be chosen. Select the configuration under “Analog – Multiple Channels – Single Sample – 1D DBL”. This version of the function will read from one or more analog input channels, and take a single sample from each, returning a 1-dimensional array of numbers of the type “double”. Since we are reading only one channel, this will produce only one data value on each call. The output from this can be wired directly to the input of a waveform chart.

The data acquisition loop also needs a way to terminate, which is the purpose of the “Stop” button. The loop should also terminate if an error has occurred, since the loop may attempt to continue executing and constantly generating errors if one were to occur. The “error out” output from the read VI can be used to determine if an error has occurred by using the unbundle function. This allows access to a Boolean value which is set to true only when an error has
occurred. Combining these two termination methods with a Boolean OR function produces the desired result.

Finally, the processing speed may need to be throttled back for this loop as well. A “Wait until next ms” function with an input of “50” should be reasonable for a simple demonstration. Also, once the DAQ loop has terminated, the task should be stopped with a simple Stop Task VI placed outside the loop and wired from the read VI

When the program is complete, it is ready to be made into an executable file. Right click the build specification seen in the project overview window, and click “Build”. If everything has been configured properly, all associated files will be saved, and then the executable is built. The process can be somewhat lengthy; a duration of several minutes is possible. Once the build has completed, copy the executable to the PDA’s memory. There are several ways to do this, but simply dragging and dropping the file using the Windows Explorer file manager is the simplest. If you’re unsure of where the executable is located, right click on the build specification used to build it, and select properties. The path of the file is displayed under “Destination directory”.
Adding Network Capabilities

A set of commands exists for communication via a special protocol, such as TCP. Technically, this set of commands is called an application programming interface (API). LabVIEW provides an API that greatly simplifies using TCP.

A TCP connection must be established before any communication can occur. Establishing this connection requires the IP address and port number of the target computer and application. If this succeeds, data can be written or read to and from the connection very similar previously discussed interfaces, such as RS-232 and GPIB. The corresponding LabVIEW functions used in this section are shown in Table 3 below, with each of their connectors’ descriptions from the LabVIEW documentation.

<table>
<thead>
<tr>
<th>Function Name and Description</th>
<th>Connector Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Open Connection</td>
<td><strong>address</strong> is the address with which you want to establish a connection. This address can be in IP dot notation or it can be a hostname. If you do not specify an address, LabVIEW establishes a connection to the local computer.</td>
</tr>
<tr>
<td></td>
<td><strong>remote port or service name</strong> can accept a numeric or a string input. <strong>remote port or service name</strong> is the port or name of the service with which you want to establish a connection.</td>
</tr>
<tr>
<td></td>
<td><strong>timeout ms</strong> is the period of time in milliseconds to wait before the function completes and returns an error. The default value is 60,000 ms or 1 minute. A value of –1 indicates to wait indefinitely.</td>
</tr>
<tr>
<td></td>
<td><strong>local port</strong> is the local connection port. Some servers only allow connections to clients that use port numbers within a specific range that is dependent on the server. If the value is 0, the operating system selects an unused port.</td>
</tr>
<tr>
<td></td>
<td><strong>connection ID</strong> is a network connection refnum that uniquely identifies the TCP connection. Use this value to refer to this connection in subsequent VI calls.</td>
</tr>
</tbody>
</table>

Opens a TCP network connection with the **address** and **remote port or service name**.
### TCP Close Connection

<table>
<thead>
<tr>
<th>connection ID</th>
<th>connection ID out</th>
<th>abort (F)</th>
<th>error in (no error)</th>
</tr>
</thead>
</table>

Closes a TCP network connection.

- **connection ID** is a network connection refnum that uniquely identifies the TCP connection you want to close.
- **abort** is reserved for future use.
- **connection ID out** has the same value as **connection ID**. Do not wire this output to other TCP functions.

### TCP Write

<table>
<thead>
<tr>
<th>connection ID</th>
<th>data in</th>
<th>connection ID out</th>
<th>bytes written</th>
<th>error out</th>
</tr>
</thead>
</table>

Writes data to a TCP network connection.

- **connection ID** is a network connection refnum that uniquely identifies the TCP connection.
- **data in** contains the data you want to write to the connection.
- **timeout ms** is the period of time in milliseconds to wait for the function to write bytes to a device before the function completes and returns an error. The default value is 25,000 ms. A value of –1 indicates to wait indefinitely.
- **connection ID out** returns the same value as **connection ID**.
- **bytes written** is the number of bytes the VI writes to the connection.

### TCP Read

<table>
<thead>
<tr>
<th>connection ID</th>
<th>bytes to read</th>
<th>timeout ms (25000)</th>
<th>error in (no error)</th>
</tr>
</thead>
</table>

Reads a number of bytes from a TCP network connection, returning the results in **data out**.

- **mode** indicates the behavior of the read operation.

<table>
<thead>
<tr>
<th>mode (standard)</th>
<th>connection ID out</th>
<th>data out</th>
<th>error out</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Standard (default)— Waits until all bytes you specify in <strong>bytes to read</strong> arrive or until <strong>timeout ms</strong> runs out.</th>
<th>Returns the number of bytes read so far. If fewer bytes than the number of bytes you requested arrive, returns the partial number of bytes and reports a timeout error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffered— Waits until all bytes you specify in <strong>bytes to read</strong> arrive or until <strong>timeout ms</strong> runs out. If fewer bytes than the number of bytes you requested arrive, returns no bytes and reports a timeout error.</td>
<td></td>
</tr>
<tr>
<td>CR LF— Waits until the function receives a CR (carriage return) followed by a LF (linefeed) within the number of bytes you specify in <strong>bytes to read</strong> or until <strong>timeout ms</strong> runs out. Returns the bytes up to and including the CR and LF. If the function does not find a CR and LF, returns no bytes and reports a timeout error.</td>
<td></td>
</tr>
<tr>
<td>Immediate— Waits until the function receives any bytes from those you specify in <strong>bytes to read</strong>. Waits the full timeout only if the function receives no bytes. Returns the number of bytes so far. Reports a timeout error if the function receives no bytes.</td>
<td></td>
</tr>
</tbody>
</table>

- **connection ID** is a network connection refnum that uniquely identifies the TCP connection.
- **bytes to read** is the number of bytes to read.
- **timeout ms** specifies the time in milliseconds that **mode** uses as the maximum time before reporting a timeout error. The default is 25,000 ms. A value of –1 indicates to wait indefinitely.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String to IP</strong></td>
<td>Converts a string to an IP network address or an array of IP network addresses.</td>
</tr>
<tr>
<td><strong>TCP Listen</strong></td>
<td>Creates a listener and waits for an accepted TCP network connection at the specified port.</td>
</tr>
</tbody>
</table>

**String to IP**

- **name**
  - STR IP
  - net address
  - **name** is the string you want to convert. If empty, **net address** is the IP network address of the current machine.
  - **net address** is the IP network address equivalent to **name**; it is the unsigned numeric representation of the dot-notation format representation of the IP network address.

**TCP Listen**

- **net address**
  - **net address** specifies on which network address to listen. Specifying an address is useful if you have more than one network card, such as two Ethernet cards, and want to listen only on the card with the specified address. If you do not specify a network address, LabVIEW listens on all network addresses. Use the String To IP function to obtain the IP network address of the current computer.
- **service name**
  - **service name** creates a known reference for the port number. If you specify a service name, LabVIEW registers the service name and the port number with the NI Service Locator.
- **port**
  - **port** is the port number on which you want to listen for a connection.
- **timeout ms**
  - **timeout ms** is the period of time in milliseconds to wait for a connection. If a connection is not established in the specified time, the VI completes and returns an error. The default value is –1, which indicates to wait indefinitely.
- **resolve remote address**
  - **resolve remote address** indicates whether to call the IP To String function on the remote address. The default is TRUE.
- **connection ID**
  - **connection ID** is a network connection refnum that uniquely identifies the TCP connection. Use this value to refer to this connection in subsequent VI calls.
- **remote address**
  - **remote address** is the address of the remote machine associated with the TCP connection. This address is in IP dot notation format.
- **remote port**
  - **remote port** is the port the remote system uses for the connection.
The Client Program

Figure 9: Existing code to which the internet connectivity will be added.

This program for the PDA is simply an extension of the program developed in the previous section, shown in Figure 9. The changes can be seen in Figure 10, with each change labeled and described below.
1. A TCP connection is opened first with an IP address specified by the user in a new string control on the front panel. Port number 13653 was chosen randomly for use with these programs. Lower numbered ports (typically 1023 and below) are well-known ports used for common services, and should not be used in a general applications such as this one. A timeout of 2000 milliseconds is given, along with a local port of 0. Wiring a 0 to this connector causes the operating system to choose a port through which to send outgoing data.

2. Since both data acquisition and data transmission occur within the inner loop, exiting this loop means that the connection is no longer needed and can be closed. Running the
Connection ID wire into and directly out of the loop ensure that the loop cannot execute until the connection is established, and will not be closed until the loop is finished.

3. Though it creates slightly more network traffic than is necessary, this program transmits one data value on each iteration. The output of the DAQmxBase Read VI contains this data value. By design, the output of the Read VI is in the form of an array, but since we are only reading from one channel, this array contains only one value. This causes a slight data type conflict, since functions used later on will not work with arrays. This is fixed with a simple use of the Index Array function with an index input of 0 to extract the one and only value in the array.

4. Now with the single value from the Index Array function, it needs to be transmitted. However, the TCP Write function requires a string data type for its “data” input. Again, the data type needs to be changed, this time from a double to a string. The “Number to Fraction String” function could be used, but the TCP read function must know exactly how many bytes to read, and since the size of a string representation of a number is not proportional to the value of that number (a value of -1.543 uses 6 characters in a string, which is equal to 6 bytes, however as a double number it uses 8 bytes to represent the same thing) some problems could ensue. The simplest solution is to simply tell LabVIEW to treat the number as if it were a string without changing any of the data. This is done with the “type cast” function. Its leftmost connector takes the data to convert, and the upper connector takes a value of the type to convert the data to (the value of this input is not used). A simple empty string constant is used here.

5. The data are in the correct format and are ready to be written to the TCP connection. The TCP Write function is used with the Connection ID, the data to write, and a timeout of
500 milliseconds. Similar to the previous project, the inner loop should terminate if an error occurs. Unbundling the “error out” from TCP Write and ORing it with the previous termination condition will accomplish this.

Once these changes are done, the executable can be rebuilt and copied to the PDA. Running this program now, however, will have little to no effect, since its proper function requires a connection to another program that will receive its data.

The Server Program

![Block diagram of the server program.](image-url)
This program continually waits for a TCP connection to be made at the previously decided port (13653), and begins reading from that port until the connection closes. This is the general model for a client-server relationship as shown in Figure 13. The data that are read are then displayed on a waveform chart and a string indicator.
Inside the main loop is the TCP Listen function. It is set to monitor port 13653 on the local network, does not timeout, and does not resolve the remote address. When a connection is detected on this port, a Connection ID is output into an inner loop, causing it to begin executing. Within this inner loop, the TCP Read function is set to read 8 bytes from this connection with each iteration. If the read function succeeds in reading, then the data must then be converted to a double number and displayed on the front panel indicators. The read function returns an error if it doesn’t succeed, which can be used as a condition.

A new case structure is created in this loop and, after unbundling the error, the Boolean value is wired to its case selector. Inside of this case structure, the data must be converted back to a double number from a string. This can be done by reversing the inputs as used in the PDA program. A string is given as data, and a double number (create a numeric constant and enter “0.0” in it) is the type to which it is converted. The output of this can be wired directly to the waveform chart, but the “Number to Fractional String” function is needed to display it on the
string indicator. To append new data to the string indicator’s text, a property node for its value can be used to read its text from another location in the program and concatenate it with the new data. A string constant of a command and a space between these allows for greater readability.

Finally, in the event that the TCP Read function fails, the loop needs to execute, since the TCP connection is mostly likely no longer valid. In addition to the “Stop” value wired to the inner loop condition, the Boolean value of the TCP Read’s error out can be compared using a Boolean OR function, with the result used as the loop condition. When this inner loop terminates, the TCP Listen function will again wait indefinitely for a connection to be reestablished.
**Experimentation**

To test this setup, the PC program must begin running first so that the TCP Listen function is listening on port 13563 when the PDA program attempts to establish a connection. With the PC program running, the PDA program can now start. After opening the program and entering the PC’s IP address, “Run” is clicked. If the PDA is connected to the same network, the data should appear on the PDA and, after a slight delay, the PC as well. This may produce a result similar to the sample below.

![Image of the PDA program acquiring, displaying, and transmitting data.](image)

*Figure 14: The PDA program acquiring, displaying, and transmitting data.*
Figure 15: The PC program listening and reading on port 13563 and displaying the data.
Conclusion

Even though it required the development of two programs (server and client), the actual coding was probably the easiest part of this project. Simply setting up everything to prepare for development was extremely frustrating, due to software incompatibilities and difficult-to-find documentation. The primary time sink, however, was the excessive compilation time and lack of debugging capabilities for the client program. Even on a relatively powerful computer, compilation of even the slightest change required 5 to 10 minutes. This was usually followed by testing which revealed yet another slight change which needed to be made.

Despite the hardships, this project has, more than anything, given me a better grasp of connection-oriented programming with TCP. As all programming was done in LabVIEW, and the target operating system was Windows-based, the coding didn’t seem much different than when the target was a typical PC.
References

