4-7-2003

Management of remote field instrumentation via the Internet

Victor A. Acuña
Florida International University

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

MANAGEMENT OF REMOTE FIELD INSTRUMENTATION VIA THE INTERNET

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

by

Victor A. Acuña

2003

To: Dean Vish Prasad
College of Engineering

This thesis, written by Victor A. Acuña, and entitled Management of Remote Field Instrumentation via the Internet, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Armando B. Barreto

M. Ali Ebadian

Jean Andrian, Major Professor

Date of Defense: April 7, 2003

The thesis of Victor A. Acuña is approved.

Dean Vish Prasad
College of Engineering

Dean Douglas Wartzok
University Graduate School

Florida International University, 2003

DEDICATION

I would like to dedicate this thesis to my parents, Blanca L. Alfaro and Victor M. Acuña, for their support and dedication towards the fulfillment of my education and goals in life. To my lovely wife, Roxana Acuña and my future son/daughter, for their love, support, and understanding in completion of my graduate coursework and this manuscript.

"Imagination is more important than intelligence." Albert Einstein

ACKNOWLEDGMENTS

I wish to thank first and for most God, for the gift of life and academic discipline. I want to thank the Hemispheric Center for Environmental Technology and Dr. Ali Ebadian for providing the economic resources and laboratory equipment to complete this thesis. A special thanks for the guidance and example of Jose Varona, Celso Duran, Leonel Lagos, Sarkis Shahin, and the rest of the Engineering and Technology Group.

I also want to extend my gratitude to my committee members, Dr. Armando Barreto and Dr. Jean Andrian, for their academic example and encouragement. My parents, Victor & Blanca, my lovely family, Roxana & baby, and my friends. Lastly, to all the great scholars of our time for their example and refusal to believe that nothing is impossible.

ABSTRACT OF THE THESIS

MANAGEMENT OF REMOTE FIELD INSTRUMENTATION VIA THE INTERNET

by

Victor A. Acuña

Florida International University, 2003

Miami, Florida

Professor Jean Andrian, Major Professor

Supervisory Control & Data Acquisition (SCADA) systems are used by many industries because of their ability to manage sensors and control external hardware. The problem with commercially available systems is that they are restricted to a local network of users that use proprietary software. There was no Internet development guide to give remote users out of the network, control and access to SCADA data and external hardware through simple user interfaces.

To solve this problem a server/client paradigm was implemented to make SCADAs available via the Internet. Two methods were applied and studied: polling of a text file as a low-end technology solution and implementing a Transmission Control Protocol (TCP/IP) socket connection.

Users were allowed to login to a website and control remotely a network of pumps and valves interfaced to a SCADA. This enabled them to sample the water quality of different reservoir wells. The results were based on real time performance, stability and ease of use of the remote interface and its programming. These indicated that the most feasible server to implement is the TCP/IP connection. For the user interface, Java applets and Active X controls provide the same real time access.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

LIST OF ACRONYMS

API:	Application Program Interface
ARPANET:	Advanced Research Thesis Agency Network
ASCII:	American Standard Code for Information Interchange
ASP:	Active Server Pages
CCC:	Central Control Console
CGI:	Common Gateway Interface
COM:	Communication ports
CSV:	Comma separated values
DOE:	Department of Energy
E&T:	Engineering and Technology group
FIU:	Florida International University
GPIB:	General Purpose Interface Bus
GUI:	Graphical User Interface
HCET:	Hemispheric Center for Environmental Technology
HTML:	Hypertext Markup Language
I/O:	Input/Output
IE:	Internet Explorer
IRQ:	Interrupt request line
JSP:	Java Server Pages
LAN:	Local Area Network
OCTOPUS:	Laboratory setup (network of pumps, valves and sensor array)
ORP:	Reduction-Oxidation

PDA:	Personal Digital Assistant
PHP:	Pearl Helper Pages
RAM:	Random Access Memory
ROM:	Read Only Memory
RPC:	Remote Procedure Call
SCADA:	Supervisory Controlled & Data Acquisition
SQL:	Standard Query Language
SRS:	Savannah River Site
TCE:	Trichloroethylene
TCP/IP:	Transmission Control Protocol/Internet Protocol
VB:	Visual basic

CHAPTER I

I. INTRODUCTION

Supervisory Control & Data Acquisition (SCADA) systems are used by many industries because of their ability to manage sensors and control external hardware. The problem with commercially available SCADAs or data loggers is that they are restricted to a local network of users that use proprietary software. There was no Internet development guide to give remote users out of the network, control and access to SCADA systems.

A. Research Problem

The thesis is derived from current work at the Hemispheric Center for Environmental Technology (HCET), a research and development center located at the Florida International University (FIU) Engineering Center. The main client is a Department of Energy (DOE) facility called Savannah River Site (SRS) located in South Carolina. This site was opened in the early 1950s to produce the basic supplies for nuclear weapons. At SRS there is a network of groundwater wells located throughout the site that must be constantly monitored for Trichloroethylene (TCE), a hazardous contaminant and carcinogen that is used as a solvent to degrease metal parts. This monitoring is done to confirm that high doses are not present in the water supplies. Usually, extraction of water samples is taken manually and processed at chemical and analytical labs offsite. This causes delay in data gathering and exposes workers to be working around the area of possible contamination. A proposed solution to this problem is to use a network of pumps to extract water from a certain well and take it to a cluster of sensors to detect different water quality parameters in near real time. There is not a

single sensor commercially available to detect TCE accurately and at regulatory levels. Many of them are still undergoing research at various universities and national labs. Thus, it was decided to measure indicative parameters of TCE such as pH, reduction-oxidation (ORP), dissolved oxygen, nitrate ions, chloride ions, specific conductance and salinity. All these are the environmental issues, but apart from these requirements users should be able to operate the network of pumps and valves from a remote location, via the Internet, and gather data on demand.

To provide an interface to drive the pumps, valves and sensors, a SCADA made by Campbell Scientific was used: the CR10X. Currently Campbell does not provide support for making web interfaces to this data logger. All software and connectivity has to be customized to this particular model and setup. Making the system available on the web brings forth many complexities and programming issues. One must understand the hardware or SCADA, learn how to program it and configure it to receive and transmit data. Software programming must be done to communicate with the hardware in an efficient manner. Internet programming needs to be used to create a user interface to run the experiment. Lastly, there must exist a link between the computer that is connected to the SCADA and an Internet application. Most of the time was spent in understanding and learning how to program the hardware and software and making the two work together.

The purpose of having access to this setup via the web is to widen the research audience and provide real time access to gather data and make an experiment. This will also lower the number of site visits, lowering costs and exposure to possible contaminated sites.

B. Experimental Setup and constraints

A small setup with five tanks to simulate wells, a transfer tank, a network of valves and a driver circuit was developed. These enabled a user to extract water from a certain tank and route it to a sensor array to sample water quality parameters. The network of pumps and valves is connected to the SCADA via a driver circuit. The SCADA communicates wirelessly to a web/application server that serves requests from different clients. The overview of the whole system is shown in Figure 1.

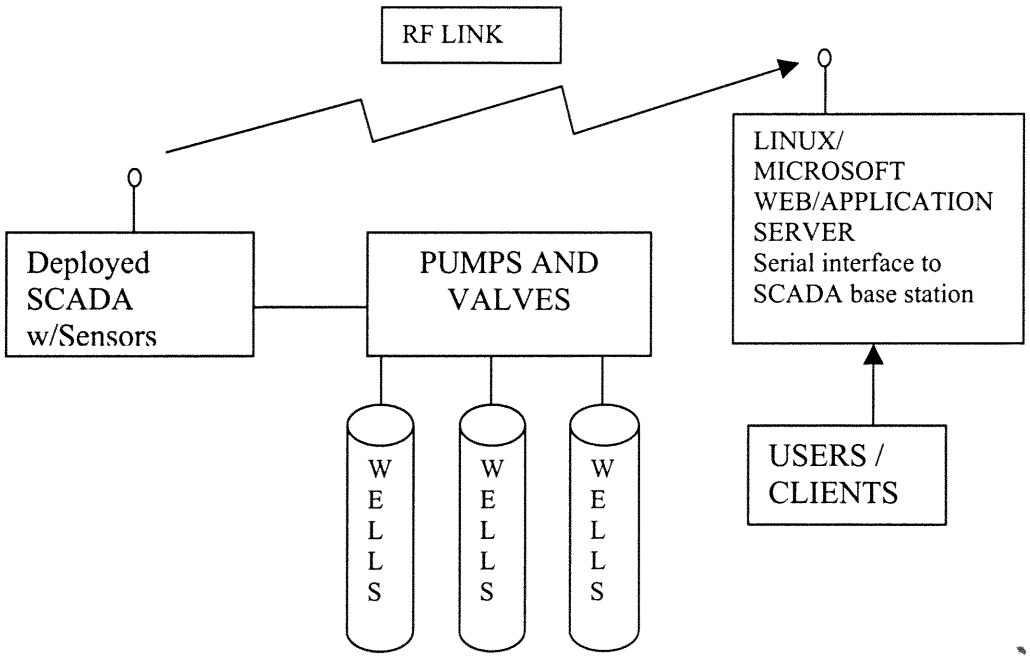


Figure 1. Schematic of bench test setup.

Figure 2 shows a close up view of the transfer tank and how water is routed to the sensors or the drain tank by the selection of valves. To perform the experiment a user will first select a tank/well to extract water from, pump it into the transfer tank until it is full, then proceed to rinse it out by opening valve #1. A second volume is extracted into

the transfer tank and then drained inside the sensors by opening valve #2, where a sample of data is taken. This water is then taken out by opening valve #3 and the sensors rinsed with fresh tap water. The process is then repeated for each different well/tank and data crosscheck to make sure that each one has different parameter readings.

Inside the transfer tank there are two level switch sensors that indicate when it is full or empty. This is used to provide the user the water level status of the tank. When the high level switch turns on and indicates that the transfer tank is full, the driver circuit disables all pumps. This feature ensures that there won't be an overflow of the transfer tank. For simplicity in writing, the network of pumps, valves and sensors will be referred to as Octopus.

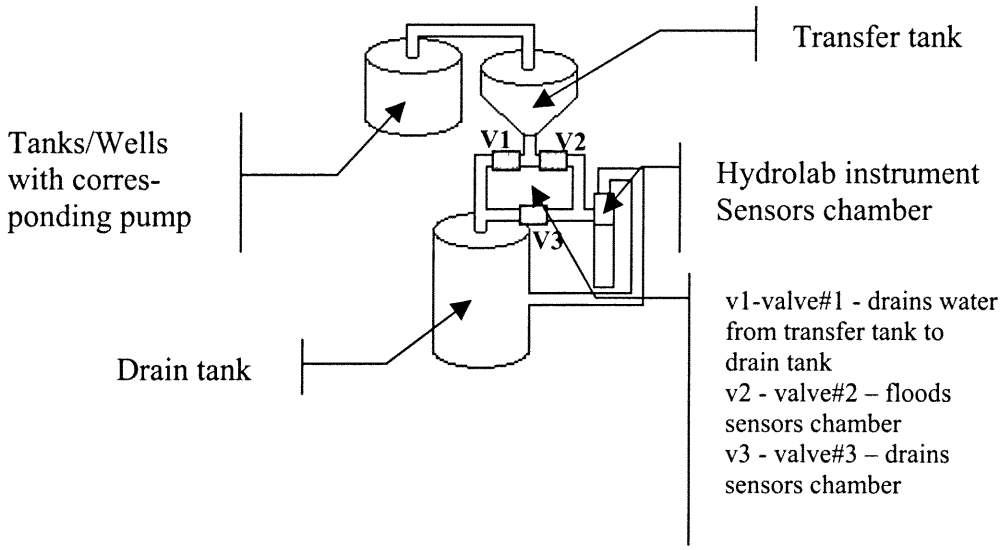


Figure 2. Close up view of pump and valve network.

The following constraints are applied to the thesis development:

1) The setup is in house because the pumps require 110VAC, thus power is provided continuously by electrical outlets (even though the setup is in house, the final

destination will be a remote field; this has to be kept in mind to allow the designs to be easily migrated towards a deployment implementation).

2) The system communicates by a wireless link, but power to the SCADA unit is provided by a power supply.
3) The development of custom software and the web interface correspond to this unique setup and the CR10X.

C. Theory of Operation

To make the experiment available via the Internet a client-server paradigm [1] was used. This is a term used in network computing to describe the state of communication of two computers. The pattern consists in one machine waiting passively for contact from another machine. The system that waits is called the server and the one that establishes contact is called the client. The transport medium used by these to communicate may be a Local Area Network (LAN) or the Internet. There are two methods studied in this thesis to permit a client-server configuration: the use of a text file as a low-end technology solution and the implementation of a transport protocol socket connection. By studying their advantages and disadvantages a final design solution will be proposed. The main parameters of interest are:

1) Performance: Can nearly real time full duplex communication be established between the laboratory experiment and a remote client?
2) Stability: Is the communication between client-server stable and reliable?
3) Client interface programming: What are the possible programming options for a web interface? Are they portable (i.e. cross browser compatible)?

The first method entails the use of a text file or database located at the web server as a global variable. In this application a text file was chosen because of its simplicity; it can be accessed fairly quickly and it takes much less memory than a database file. Not to mention that a String Query Language (SQL) server is not needed, which would be the case if a database file was used. Internet server scripting can have access to this file as well as a software applications running on the same machine. Data can be interchanged between the web server and the local application creating a transparent link between a remote user and the laboratory experiment. The local application will have a server module that has a direct or wireless connection with the SCADA unit and polls the text file for accepted commands. This is the same concept of accessing a public variable in a computer program. If it finds a command that it understands it will execute the desired function, otherwise it will be ignored.

The second method uses a transport protocol socket connection between the server and the client. The two protocols considered were the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Each one has advantages and disadvantages, but can provide communication between the client and the server. The transport protocol server is setup to listen or wait at a specific port on the web server for a connection request. When the client creates an authorized socket connection at the same port that the server is listening to, a link is created and data can be interchanged between the two machines. The client then passes commands to the server. The server passes the commands to the main program. If a command is recognized it will be executed, otherwise it will be ignored.

There are three stages of programming for these two methods: the actual hardware, the remote server and the website. In order to explain how this was accomplished, different issues will be discussed, those include embedded systems, serial communications, and Internet programming. Instead of introducing each concept and explaining the overall system at the end, the actual timeline of development (results and observations) will be migrated in the discussion of each chapter. The first prototype will be tested with the experimental setup.

The stages of development are outlined in Figure 3:

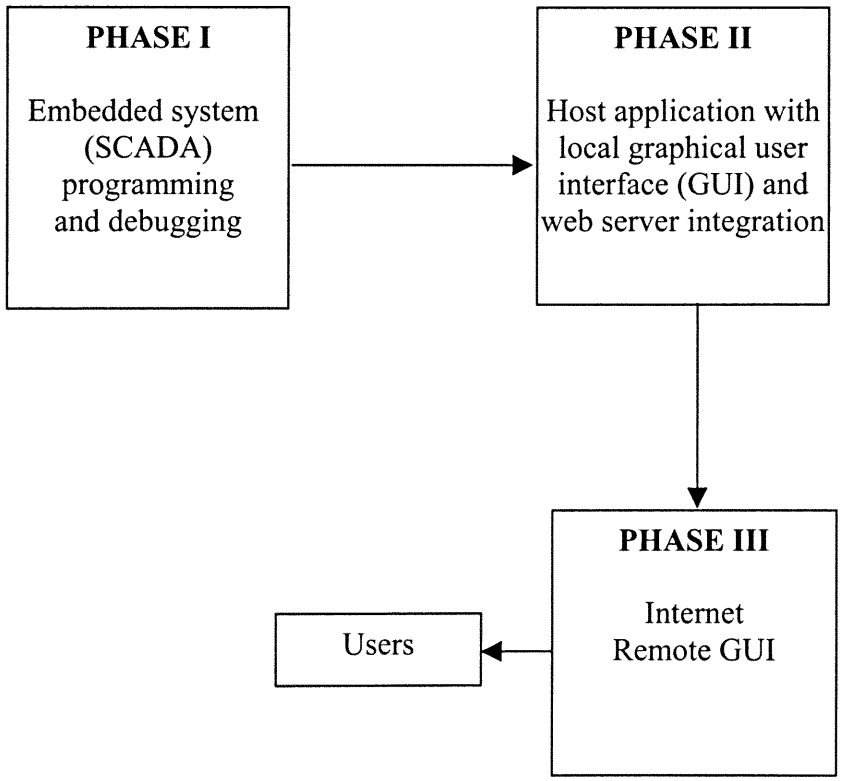


Figure 3. Thesis development timeline.

D. Literature Review

To provide a solution for remote access via the web a literature and web search was done. The results were that a complex web site equipped with management capabilities of the CR10X SCADA systems is not available. Thus to do this, a process must be developed and all software platforms would have to be programmed and customized for the desired application.

Many researchers have embarked in the field of remote control of laboratory experiments via the Internet. They all use different processes and technology, but none are for the CR10X and most of them are used for learning/educational experiments [2, 3, 4, 5 and 6] instead of a deployable solution for a remote site, much like SRS. There have also been a relative small number of real time experiments and much focus on non real time virtual laboratories [7].

Remlab [2] is a remote measurement laboratory for educational experiments. The processes used in [2] are not interactive or dynamic. They use Visual C++ to create an application program that communicates with the hardware and sends Hyper Text Markup Language (HTML) back to the user via the Common Gate Way (CGI) interface method. When a client initiates a request, a C program is executed that performs the function and then returns the result to the browser (i.e. Netscape or Internet Explorer (IE)).

Peter Hoo [8] worked in the area of Internet sharing of General Purpose Interface Buses (GPIB). GPIB is used to handle data acquisition hardware to make measurements. He was able to excite sensors and gather data to make it available through a web page. He uses the same method as [2]. The problem with this kind of method is that it works well for a page that has a couple of lines, but for complex sites that require large amounts

of code to manage hardware and process data, this is not a feasible solution because it makes the maintenance of the website cumbersome. This also means that the website is not interactive and independent from the host programs that access the hardware. Status of the system is done by user requests because the devices are not smart enough to transmit data back constantly like a SCADA.

[3, 4, 5 and 6] are more advanced remote laboratories where client-server paradigms are implemented via transport protocols, mainly TCP/IP. In their implementations, the client software creates a socket connection to the server and uses a client Graphical User Interface (GUI) to perform experiments. The use of this method is innovative, but requires the client to download client software and install it in the remote machine. It is desired that the client-server communication be established through a web browser (i.e. Netscape or Internet Explorer) because they are easily accessible in all operating systems. Access methods via a browser makes it easier for remote users to access the experiments and collect data from any computer in the world, without the need of downloading and running client software.

An ideal way for remote communication with hardware is to develop a set of functions so that a developer can create an interactive web site that communicates with the hardware without the need of knowing the background intimacies of the program that controls the hardware. The "Remote Laboratory" introduced in [9] made a significant change in the design of remote control experiments. A socket connection was used to communicate with the server via a client implemented in a Java applet. Applets are small container programs that are downloaded and executed inside a web browser. This eliminates the use of a separate client software package. Although [9] uses Java applets,

they did not venture into using Active X controls. Active X are like applets, they execute inside a web browser and can be programmed to provide advanced features to the client.

Some SCADA systems provide LAN based configurations so that they can be accessed via a network. [10] Used LAN based control of SCADAs to monitor load shedding in power systems. This is acceptable for systems that are not field deployable. The initial work on this thesis is for an in house experiment. But the final destination of the experiment is to be a remote location, where a LAN may not be accessible, only serial communications via a wireless modem.

The demand to have user access to remote devices via the web did not become popular until recent years [11] and that is why there is not a definite development platform to get this done. Not to mention that every application is different and that is why many solutions have to be studied before proposing a final implementation. None of the authors considered the polling method of a text file as a low-end solution. The work in this thesis proved that this could also be a viable solution when there are not enough programming resources available.

Campbell Scientific does not provide Internet applications support, only data management for online display [12]. This becomes more of an issue of displaying a database of values on a web page. That is why all software platforms and client integration had to be customized.

This review demonstrates the efforts of remote control of instrumentation mainly for learning experiments at different universities and institutes. Some lack portability and others contain full web deployment capabilities. Although none of them work with a

SCADA similar to a CR10X, different methods can be used to provide a solution to make this system available via the Internet.

CHAPTER II

II. EMBEDDED SYSTEMS

The first issue before going live on the web is to understand the final link in the connection setup, the hardware.

Embedded systems are basically tiny computers with microprocessors built for a specific purpose. They are found in many of today's electrical equipment, such as cameras, Personal Digital Assistants (PDA), cars, consumer electronics, etc. They differ from a full-blown computer in that they are small, requiring small amounts of power and that they load programs from Read Only Memory (ROM) instead of Random Access Memory (RAM) as workstations do. SCADAs are essentially embedded systems or micro controllers composed of internal memory, a microprocessor, I/O ports and a transmission media. Their purpose is to manage sensors, data and external hardware. Selection of this type of hardware depends on certain factors:

1) Capabilities: what can they do? Including computing power, I/O ports quantity and remote deployment flexibility.
2) Data management: some rely on proprietary database structures like SQL and others manage data as Comma Separated Value (CSV) text files.
3) Adaptation to custom software.

The data-logger chosen for this application was a Campbell Scientific CR10X for the following reasons:

1) Data can be extracted as CSV values that can be displayed in any type of data management software like Microsoft Excel or Lotus Qpro. CSV files also tend to

be small in size since they are made up of plain American Standard Code for Information Interchange (ASCII) characters.

2) Small programs can be written and uploaded to the unit to perform a vast amount of functions and commands.

3) It contains several digital and analog I/O ports, as well as SDI-12 interfaces to handle different types of sensors.

4) It only requires 12Volts DC of battery power and can also be operated by a solar panel.

5) The adaptation of custom software to communicate with it can be easily accomplished since is all done via serial communications.

A. Programming of embedded systems

Embedded systems programming is usually done with a type of assembly language developed for the specific device. The language of the CR10X is a set of instruction codes that perform several high level language functions such as if, then, loop, goto, store, delete, compare, etc. There are a total of 123 instructions: 49 for input/output, 39 for processing, 22 for program control and 13 for output. They can be used to automate the SCADA or to give it some user interaction. All the instruction codes and their functions are found in the operations manual. To understand the system, this manual was studied thoroughly to explore all the capabilities of the system, and the different ways it can achieve a certain function.

The CR10X has three tables that can hold program code. Table one and table two run programs at certain scan intervals, meaning that the sequential code will be executed after the time that was specified at the beginning of the program. Two main programs

were developed to make the system interactive. Program one runs every minute and checks for a status flag. A flag is just a value stored in a memory location that is used as a qualifier. If it is set to true then it proceeds to trigger the sensors and obtain a data string from them, putting them in intermediate storage for program two to process. Program two runs every ten seconds and sends a serial ASCII dump to the host computer. This string of characters contains the parameters read by the sensors and the status of the level switches located inside the transfer tank. These two programs were sufficient to make the system interactive with a remote user. The program flow is shown in Figures 4 and 5.

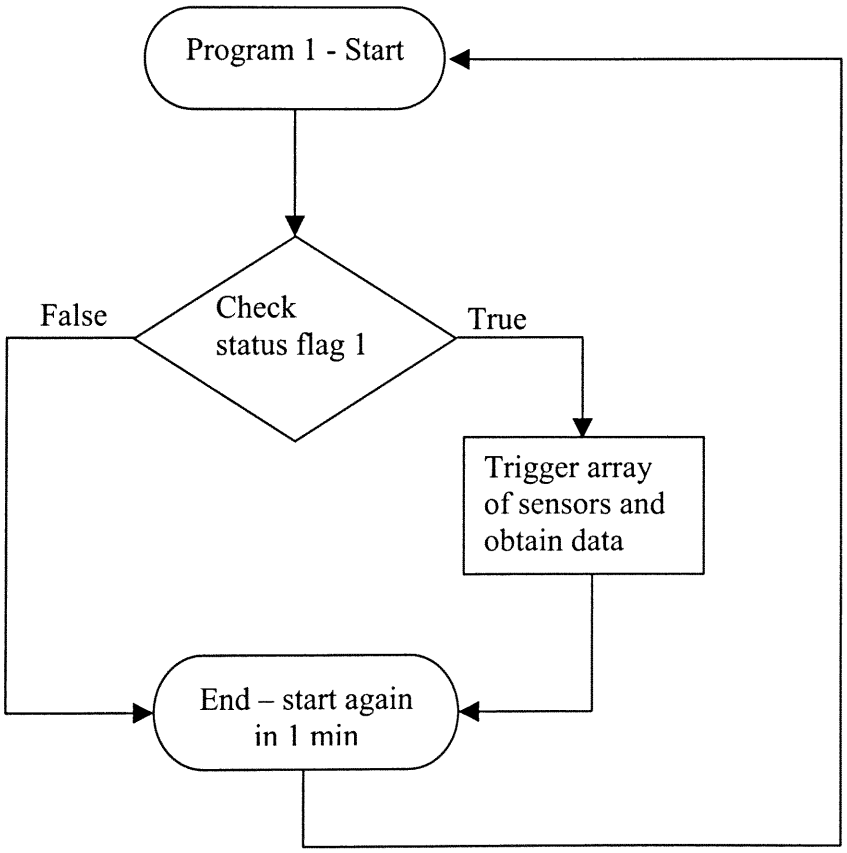


Figure 4. Program 1 (triggers sensors when told to do so).

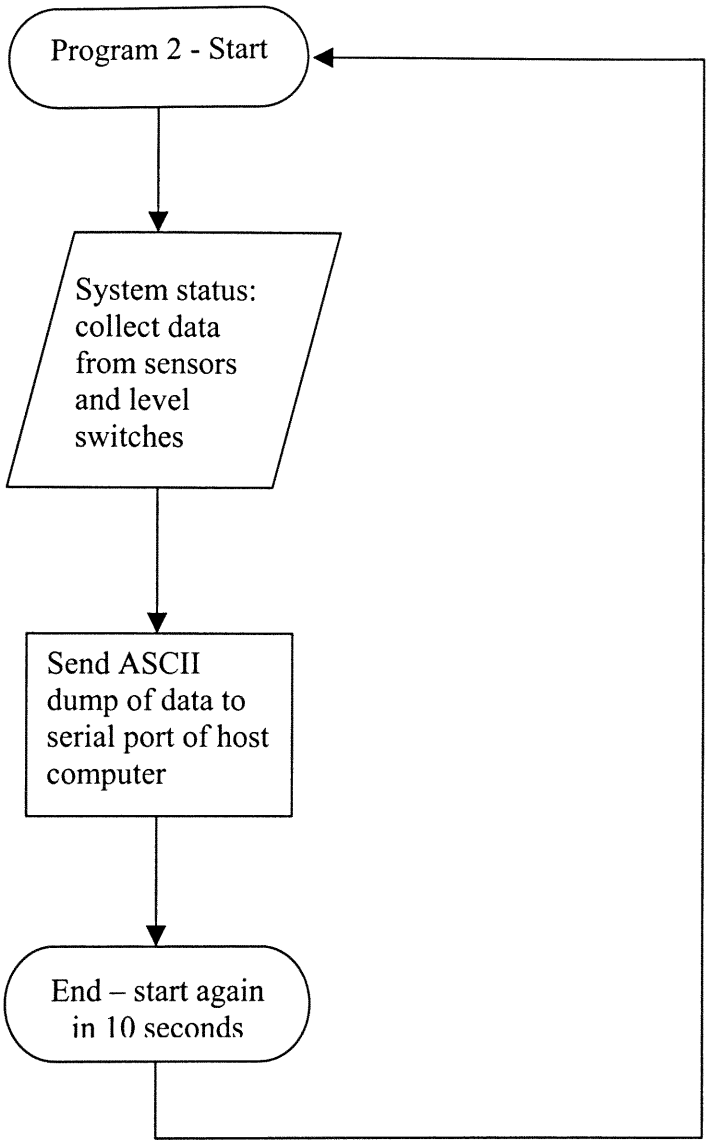


Figure 5. Program 2 (sends status data back to host computer).

B. Interfacing the SCADA with external hardware

The CR10X has 8 digital I/O ports that can be used to access external hardware when configured as outputs, they can be set high (output 5V), set low (output 0V) and apply a voltage pulse. This gives the flexibility of driving digital circuits perfect for this experiment. Since there are a total of 5 pumps, 3 valves, 2 float level switches and 1 SDI-12 sensor array in the laboratory setup, there wasn't enough room to accommodate all the hardware. The device cannot handle high levels of currents either, which was needed because the pumps are 110VAC. To overcome this problem, a 3-8-decoder driver circuit was designed to drive all the peripherals one at a time. The decoder would be driven by the CR10X and will only require 3 I/O ports. Two other I/O ports will be used for the low and high level switches to indicate when the transfer tank is empty, full or with water. The last I/O port is used for the SDI-12 interface to the sensors. SDI-12 is a serial communication standard for transducers that requires a single data channel for a cluster of sensors. In this setup, an instrument called the Hydrolab DataSonde 4 was used, it is equipped with several sensors to read pH, ORP, salinity, temperature, dissolved oxygen, nitrate and chloride. The SDI-12 permits all data from these sensors to be sent by a single data string through one channel, thus a data set per sensor is not required saving the use of many I/O ports. Figure 6 shows how the SCADA is interfaced and Table 1 provides an outline of the ports and their functions.

The CR10X set of instructions is a series of numbers that consist of a main instruction number followed by sub parameters that behave as different properties of that particular instruction. For example, to write a line of code that tells the CR10X to make

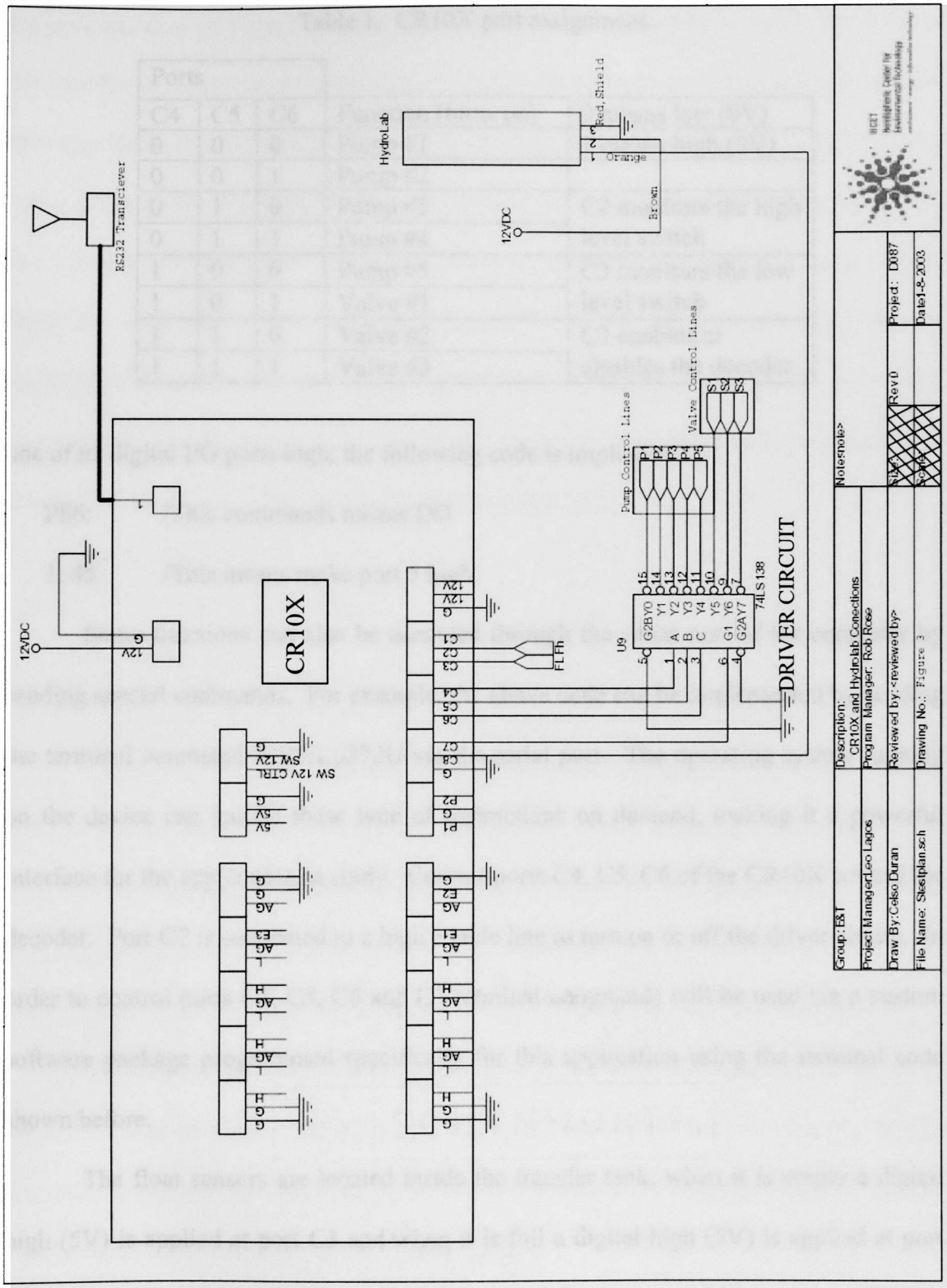


Figure 6. CR10X and driver circuit (courtesy of HCET E&T).

Table 1. CR10X port assignment.

Ports				
C4	C5	C6	Function (turns on)	0 means low (0V)
0	0	0	Pump #1	1 means high (5V)
0	0	1	Pump #2	
0	1	0	Pump #3	C2 monitors the high level switch
0	1	1	Pump #4	
1	0	0	Pump #5	C3 monitors the low level switch
1	0	1	Valve #1	
1	1	0	Valve #2	C7 enables or disables the decoder
1	1	1	Valve #3	

one of its digital I/O ports high, the following code is implemented:

P86: /This commands means DO

1: 45 /This means make port 5 high

Some functions can also be executed through the serial port of the computer by sending special commands. For example, the above code can be implemented by sending the terminal command 9105:1:0372U via the serial port. The operating system running on the device can handle these type of instructions on demand, making it a powerful interface for the application in study. Control ports C4, C5, C6 of the CR10X control the decoder. Port C7 is connected to a high enable line to turn on or off the driver circuit. In order to control ports C4, C5, C6 and C7 terminal commands will be used via a custom software package programmed specifically for this application using the terminal code shown before.

The float sensors are located inside the transfer tank, when it is empty a digital high (5V) is applied at port C3 and when it is full a digital high (5V) is applied at port C2. A program running inside the SCADA checks for the status of this ports, when one goes high it sends a signal stating which specific ports went high. This is implemented in

the program flow of Figure 5 in the data section. It was necessary to know the status of the transfer tank because the user needs to know if it is empty or full to proceed with the next step in the lab experiment. To accomplish this, instruction 25 was used to check the status of the ports. When this instruction is used, it creates a mask of the specified port. For example to check port C3, the eight ports are treated as an eight bit binary number from C8.....C1, where C8 is the most significant bit and C1 the least significant bit. By masking decimal 4, where the binary equivalent is 0000 0100, the CR10X checks this port only (C3), if the port is high it writes the number 4 in the input location specified, if is low, it writes a zero. The code to do this looks like this:

```
1:  Read Ports (P25)
 1: 0004     Mask (0..255)
 2: 23       Loc [ LOWLVLSW  ]

2:  Read Ports (P25)
 1: 0002     Mask (0..255)
 2: 24       Loc [ HIGHLVLSW ]
```

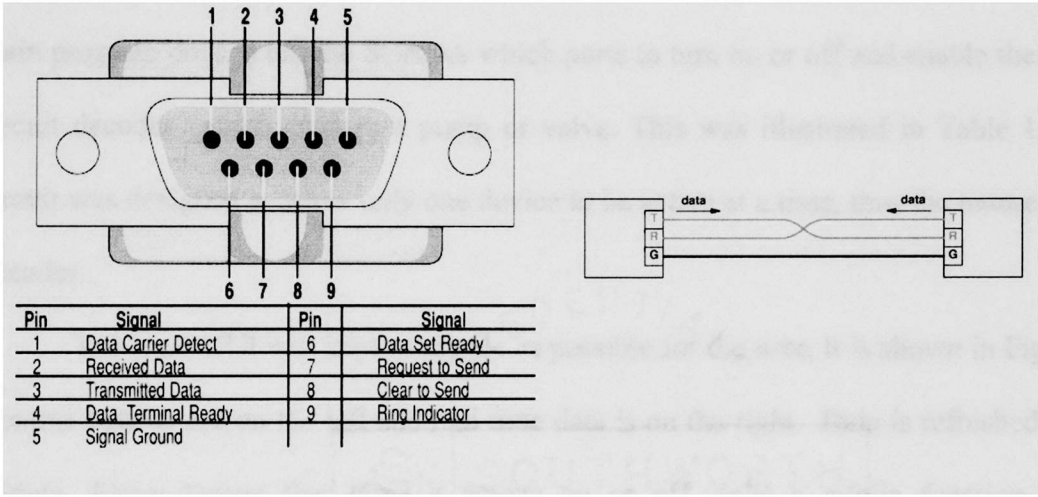
These two values are then appended to the data of the sensors and the status of the level switches can be read. If a 4 is transmitted it means that port C3 is high or the transfer tank is full, if a 2 is transmitted it means that port C2 is high or the transfer tank is empty, and if a 0 is transmitted per sensor, then there is water present in the transfer tank. There are only 3 states allowed, so if one port is high the other one must be low.

At this point the assembly programs were kept simple since the objectives of the bench test was to obtain real time data when the experiment was in progress and to determine if the transfer tank is empty or full to rinse water volumes through it. This was also the only data necessary for the web interface.

CHAPTER III

III. SERIAL COMMUNICATIONS

Communications with hardware in a computer system can occur through different ports provided in the main board. In this thesis the port of main interest is the serial port, since most SCADA systems provide a means of serial communications. A typical layout of a serial (RS232) and its data transfer signals is shown in Figure 7.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure 7. RS232 (DB9) pin out and transmission scheme.

The COM ports in a computer can access serial interfaces and a unique IRQ is assigned to it so that any device attached does not conflict with any other COM ports. Devices connected to the serial port follow unique protocols that understand the character set sent between it and the user interface. Access to the COM ports can be done with any high level language with native methods (Visual basic, Java, C, C++, etc).

A. Main program

Before going live on the web and programming the web interface, one has to know what functions or commands the hardware will understand. The server is a module added to the main program used to control the hardware setup and the SCADA. When the server module is enabled, Internet users will have access to the hardware setup.

Visual basic (VB) was used to create a local GUI for the operator to control Octopus. VB has functions to access the serial port to send and receive data. What the main program does is tell the SCADA which ports to turn on or off and enable the driver circuit decoder, which enables a pump or valve. This was illustrated in Table 1. The circuit was designed to allow only one device to be active at a time, thus the nature of the decoder.

The local GUI was kept as simple as possible for the user, it is shown in Figure 8. Control buttons are on the left and real time data is on the right. Data is refreshed every minute. Every button that turns a device on or off, calls a public function called "send(data)." This function sends data to the serial port, which is opened at execution time. Data is sent to the CR10X as ASCII character strings followed by a carriage return or character 13. An ASCII character is the 7 bit representation of each character in a computer keyboard, for example the carriage return is character 13 in decimal notation and 0D in hexadecimal.

A user can record data by clicking on the button "Record Data" and check the status of the water levels of the transfer tank. The two big white windows are mainly for debugging, but in the snapshot of Figure 8 it can be seen how raw data is transmitted between the CR10X and the main program. There is also a button called "Server Mode,"

when clicked it will start the web server so that a remote user can have access to the experiment.

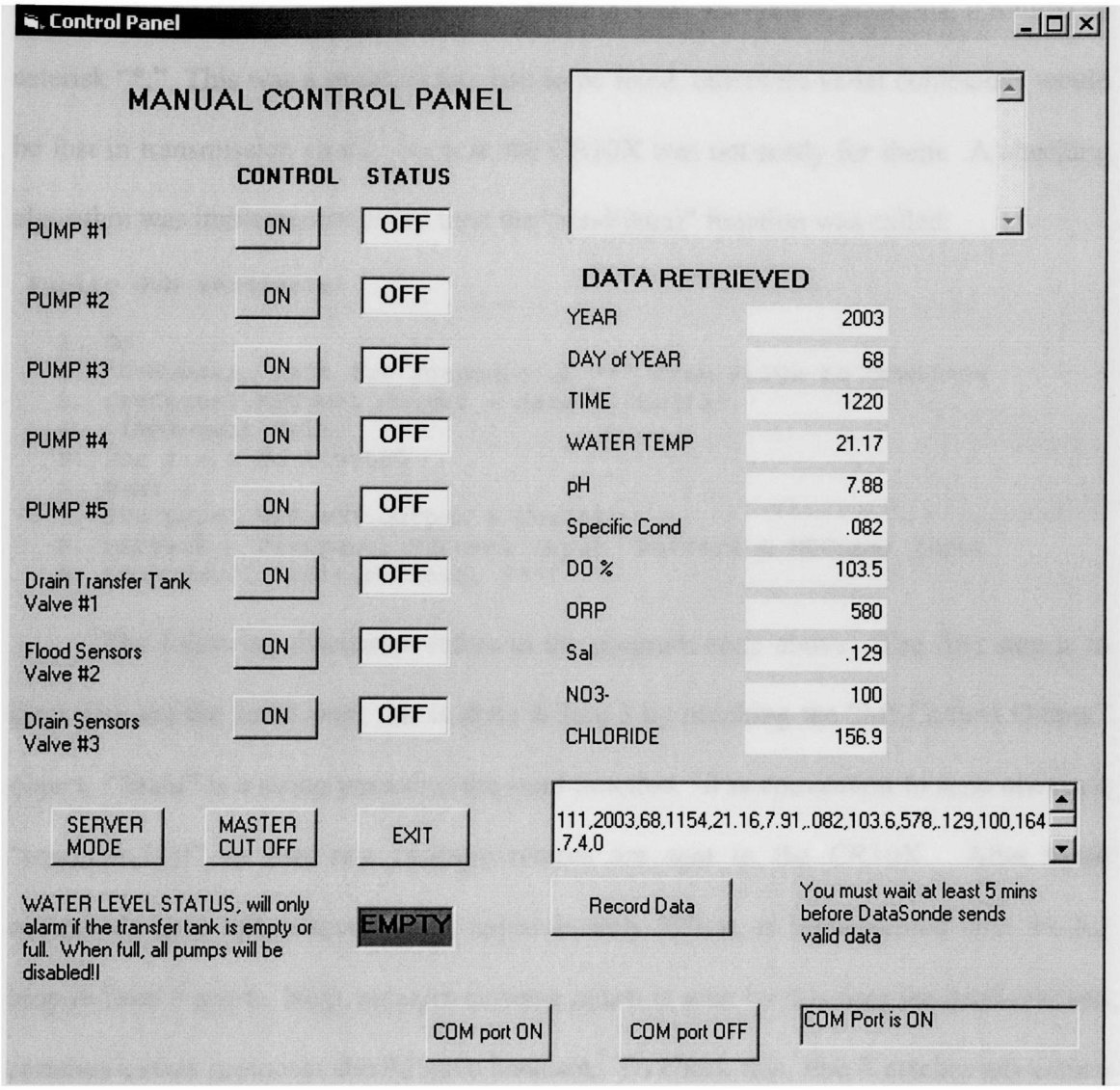


Figure 8. Snapshot of local/main control panel.

Another issue encountered is that the CR10X goes to sleep mode after 40 seconds if the ring line of the serial port is not high. This line is set high by an external device connected to the other end of the serial cable when it initiates a call. Before any

commands can be sent to the SCADA, it must be awaken. To do this, several carriage returns need to be sent so that the baud rate and communication protocol is established between the pc host and the instrument. When is ready for communications, it returns an asterisk "*." This was a problem that had to be fixed, otherwise serial commands would be lost in transmission simply because the CR10X was not ready for them. A checking algorithm was implemented every time the "send(data)" function was called:

```
Public Sub send(data)

1. Do
2. 'DoEvents, wait for response of "*" from cr10x to continue
3. frmCpanel.MSComm1.Output = data + Chr(13)
4. 'implement delay
5. For i = 1 To 1000000
6. Next i
7. frmCpanel.MSComm1.Output = Chr(13)
8. buffer$ = frmCpanel.MSComm1.Input 'Buffer$ & MSComm1.Input
9. Loop Until InStr(buffer$, "*")
```

The following discussion refers to the program code above. The first step is to send data out the serial port, this is done in line 3 by invoking the "MSComm1.Output" object. "Data" is a string passed to the send function. It is convenient to send always a "send(chr(13))" so that two carriage returns are sent to the CR10X. After these commands have left, a small delay, approximately 300ms, is implemented with the for loop in lines 5 and 6. Next, an extra carriage return is sent; by this time the baud rate and communication protocols should have been set. To check this, line 8 catches any return string with the "MSComm1.Input" object and stores it in the buffer. The buffer is a temporary memory location that the serial port uses to store incoming strings. The buffer was set to receive only a byte at a time, this type of programming ensures that each incoming character is checked one at a time. Line 9 checks to see if the buffer has an "*," if it doesn't it returns back to line 1 to start the loop and send carriage returns to get

the SCADA out of sleep mode. If it does find an "*," it proceeds to the next set of instructions. An example of how this algorithm checks for communication stability, is shown by following the sequence of turning pump 1 on:

```
1. send (chr(13))
2. send ("9104:0:0370U")
3. send ("9105:0:0371U")
4. send ("9106:0:0372U")
```

Before any commands are sent, a carriage return (chr(13)) has to be transmitted to wake up the SCADA. The program waits for confirmation that it is ready to receive additional commands. If the previous algorithm had not been implemented, the string of data would have been sent one after the other and some of them might gotten executed and some perhaps not. When confirmation is received that the CR10X is ready to communicate, the second command in line 2 is sent. This turns port C4 low, the delay is implemented and the main program waits for another "*" meaning that the command was sent successfully. Then the other two commands are sent one at a time. All ports C4, C5 and C6 should be low at this time enabling pump #1. The algorithm assures that each command is executed one at a time to avoid any undesired function.

Another problem in programming the local software was that data strings were being lost because of data transmission speeds. Strings of data received by the local program started to fill up the buffer and write on the previous data before this one was processed and displayed. In order to avoid this, the buffer length was limited to a byte or one character and the following algorithm checked the incoming strings to make sure they were correct.

```
1. Private Sub MSComm1_OnComm()
2.     Dim strDataRaw As String
3.     'store incoming data
4.     strDataRaw = frmCpanel.MSComm1.Input
```

```
5.    If headcount = 3 Then
6.    If strDataRaw = Chr$(10) Then
7.    'split comma separated values to get individual values
8.    strTempdata = Split(indat, ",")
9.    If UBound(strTempdata) = 13 Then
10.   headcount = 696
11.   Else
12.   indat = ""
13.   headcount = 0
14.   End If
15.   End If
16.   indat = indat & strDataRaw
17.   End If
18.   If strDataRaw <> "1" And headcount>0 And headcount < 3 Then
19.   headcount = 0
20.   End If
21.   If strDataRaw = "1" And headcount < 3 Then
22.   headcount = headcount + 1
23.   End If
24.   'debugging windows
25.   txtData.text = txtData.text & strDataRaw
26.   'data must be filtered before writing
27.   If headcount = 696 Then
28.   "Program Code Continues here"
```

The CR10X was configured to send a string of characters similar to this: 111,2003,64,2115,24.42,7.05,.304,103.9,683,.134,6.6,5.21,0,0. The order of this data and the meaning of its contents is outlined by table 2:

Table 2. Data order and outline.

Order	Parameter	Value
1	Year	111
2	Date	2003
3	Julian day	64
4	Military time	2115
5	Temperature	24.42
6	pH	7.05
7	Specific conductance	0.304
8	DO%	103.9
9	ORP	683
10	Salinity	0.134
11	Nitrate	6.6
12	Chloride	5.21
13	Low level switch status	0
14	High level switch status	0

Data is sent in this fashion by the CR10X every 10 seconds to have a constant string available by the main program and so that the delay of an event would be small. The algorithm checks the starting set "111" that represents the memory location of the CR10X, then it builds the string until a line feed is attached at the end of transmission. It then checks to make sure is made up of 14 elements, then it will extract the data for display and process the status of the float level switches to determine if the transfer tank is full, empty or in between. To accomplish this every byte that comes in is first compared with the number "1", line 21 checks for a "1" and line 22 increments the variable "headcount" by one. When "headcount" is 3, meaning that "111" has been received, it then starts to build the string of data. If a line feed, ASCII character 10, is received the built string is compared to the corresponding size of elements, in this case 14. If it is true then this data is taken for processing, otherwise it is ignored.

This type of processing is very important because the real time data and status of the system must be available without errors. Once the main program is working ok, meaning that data transmission works fluently, then the server can be designed.

B. Server programming

The client-server paradigm is used to create server modules to communicate with a remote client. The client and server are involved in a state of communication [1]. The server waits for the client to initiate contact. When communication is established, data can be interchanged between both services. Figure 9 illustrates the concept.

The client application has the following characteristics:

1) Becomes a client temporarily when remote access is needed.
2) Runs on the client's computer.

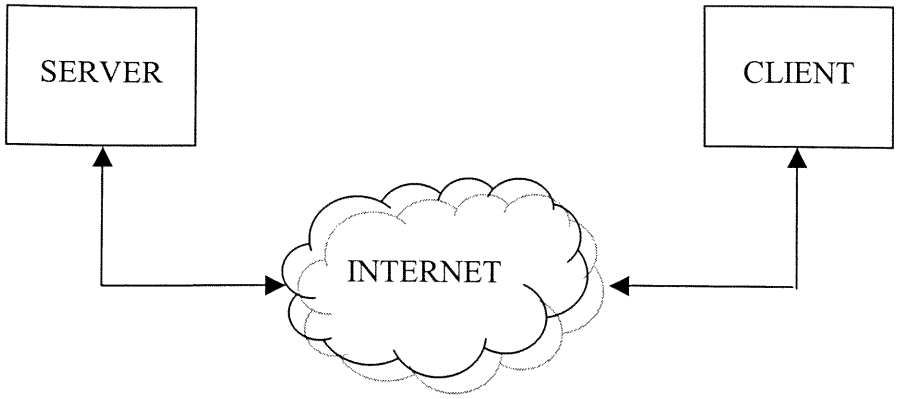


Figure 9. Client-server paradigm

3) It can be executed in different operating systems.

4) It is initiated by the remote user

The server application has the following characteristics:

1) Waits for contact from a remote client.

2) It usually resides in a more sophisticated computer with greater resources than the client.

3) It is dedicated to provide one service, but can serve multiple clients.

The Internet was not originally designed for controlling remote hardware, but more for the sharing of information, text, audio and video. During the years though, enhancements in Internet technology and the need for remote management have made this a popular area of research [11]. There are three reasons why this is so:

1) The Internet is platform independent, meaning that viewers can see the same web page no matter what operating system they are using (i.e. Linux or Windows).

2) All workstations contain a browser (a program to access web pages on the Internet, like Internet Explorer or Netscape) installed. This makes the sharing of information and small programs attractive because users don't need to be running specific third party software to access web sites and see their content.

3) When all devices are connected, a virtual component network [11] is created and all devices share common resources.

1. Text file server

The text file server module is a timer subroutine that polls a text file every second. It reads lines found in it sequentially, if it finds a command that the function understands it passes it for processing to the "turnAction()" function, otherwise it does nothing. To accomplish this a timer event function was created. This subroutine executes every second:

```
1.  Private Sub Timer1_Timer()
2.  Open "e:/downloads/rms_srs/rms_srs_local/server.vtc" For Input As
    #1
3.  Dim text As String
4.  Do Until EOF(1)
5.  Line Input #1, text
6.  If text = "" And EOF(1) = True Then
7.  Close #1
8.  Exit Sub
9.  Else
10. lstStatus.AddItem text
11. turnAction (text)
12. End If
13. Loop
14. Close #1
15. Open "e:/downloads/rms_srs/rms_srs_local/server.vtc" For Output
    As #1
16. Print #1, vbNullString
17. Close #1
18. End Sub
```

Line 2 opens the file "server.vtc" for reading; this file should reside in the same directory of the web site. If it finds a blank line and is at the end of the file it exits the function and remains idle until the next second, this is done in line 6. If it finds a command, it is passed to the function "turnAction(text)" in line 11. This function checks to see if the text written matches with one of the authorized commands. If the command is authorized it is executed, otherwise it is ignored. When the processing of the function is done the text file is closed. A piece of the function "turnAction()" shows that if it finds the string "000" it will send commands to the SCADA to turn ports C4, C5 and C6 low via the "send(data)" function.

```
1.  Public Sub turnAction(strCmd1 As String)
2.  Case "000" ' select decoder
3.  send ("9104:0:0370U")
4.  send ("9105:0:0371U")
5.  send ("9106:0:0372U")
6.  Case Else
7.  Exit Sub
```

When the timer function finishes reading from "server.vtc," it deletes all lines and writes a blank character, so that when it is not in use, it remains idle waiting for another command.

The uniqueness about this method is that it separates the website from the hardware conflicts. The functions can be given to a web developer so that he knows what commands to write to the text file to perform a specific function, but he won't have to deal with all the issues of serial communications, programming the SCADA and the main program. This also permits flexibility in maintaining a web site without interfering in program operation. Any main program that is used to access a serial device can be setup to go to server mode and use the original functions to handle requests from an

outside source. When in server mode, a local user cannot access Octopus, and vice versa, when in local mode a remote user cannot have access. This avoids problems with many users trying to access the setup at one time.

It is secured because the commands sent between a web browser and the server are codenames that have nothing to do with the actual execution codes that the CR10X understands. This means that if a hacker intercepts transmissions, all they will see is codes like 000, 111 or OK, which don't mean anything useful.

2. TCP/IP server

The transport protocol server uses TCP/IP to make a connection with a client. TCP/IP is a connection-oriented protocol. This means that the server must wait for contact by the client. When it accepts the request, a socket connection is created between the two. The TCP/IP server module designed serves only one request at a time. This is done intentionally so that there is only one user accessing the setup at a time.

Implementing a TCP server is advantageous because it provides stability and reliability [1]. The service offered by TCP has the following major features:

1) Connection Orientation. A client must request a connection first before any data can be transferred.
2) Point to point communication. There are only two endpoints in a communication link.
3) Complete reliability. TCP guarantees that the data sent will be received without errors.

4) Full duplex communication. Data can flow in either way without affecting incoming or outgoing commands. It uses input and output buffers to maintain a continuous flow of data.

5) Stream interface. TCP sends streams of octets across a network, where they will be build up at the destination to form the original message.

6) Reliable connection startup. The client and server must agree on a connection per session. Whatever happened in the last session is forgotten.

7) Graceful connection shutdown. Data is guaranteed to arrive at its destination before a request for closing the connection is sent.

All messages involved in a TCP communication follow a particular segment format [1] as illustrated in Figure 10. When the client sends a message, it is partitioned in different segments or packets. The server receives all these packets and must reconstruct the original message. The different parameters are crosschecked by the receiving application to make sure it understands what the client is requesting. Every segment contains the following information:

1) Source port. The port number of the client.

2) Destination port. The port number of the remote server.

3) Sequence number. This specifies the sequence of the data sent in this particular segment.

4) Acknowledgement number. Specifies the sequence number of the data received.

5) Window. Specifies how much buffer space is available for data.

6) Checksum. The checksum that covers the header and data.

Communication is established via the use of socket communications. Sockets were originally part of the Unix I/O, and they follow a open-read-write-close paradigm. Many details must be specified before creating a socket connection, these include: the transport protocol, host address of remote machine, port number and specify if the application is a client or server.

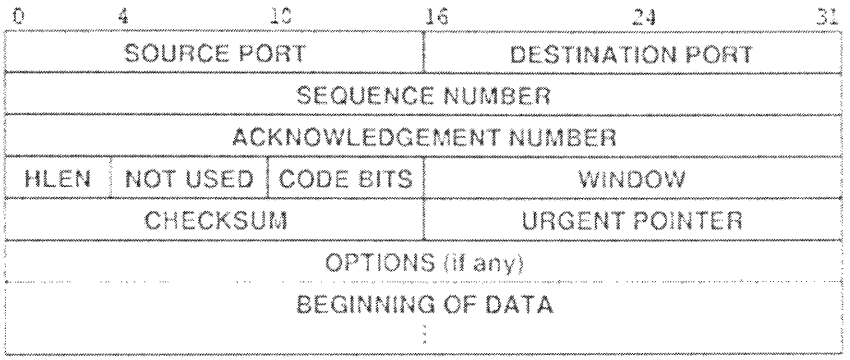


Figure 10. TCP segment format.

Before a client application can connect, the server must be listening or waiting at a specific port for a connection request. Different software platforms offer the socket Application Program Interface (API). To build the TCP/IP server, VB offers the Winsock control socket API. Since the application is a server, it must be set to listen at a particular host name and port number. The host name is "tdid-4argf.hcet.fiu.edu" and the port used is "1001." The port was chosen arbitrarily, as long as it is not any of the reserve ports of the server computer (i.e. port 80 handles the HTML requests). When the server is started it must be set to listen by invoking the "TCPServer.listen" method (see Appendix II, TCP server).

There are four major components of the Winsock control that handle all communications:

1) TCPServer_DataArrival (ByVal bytesTotal As Long): handles incoming data.
2) TCPServer.senddata: sends data.
3) TCPServer_ConnectionRequest (ByVal requestID As Long): accepts incoming connection requests. If the host and port fields are correct (i.e. host: tdid-4argf.hcet.fiu.edu, port: 1001) the connection is accepted.
4) TCPServer.close: closes the socket connection.

The TCP server receives data from the client and passes it to the "turnAction()" public function. This function verifies if the command is correct to initiate hardware communication. Status of the lab setup is sent by the TCP server via the "TCPserver.senddata" method. The client receives this data and processes it for display purposes. When the remote user finishes with the experiment, the "TCPserver.close" method is called to close the socket connection and the server returns to listening mode.

This method is also secured because the commands sent between a web browser and the server are codenames that have nothing to do with the actual execution codes that the CR10X understands. This feature was observed in the text file server.

CHAPTER IV

IV. INTERNET PROGRAMMING FOR TEXT FILE SERVER

The Internet, which means internetworking of computers, has become almost a necessary tool in everyday life. Its uses range from seeing real time stock data from Wall Street, to checking emails, reading news and using web cams for teleconfering. It began as a research program hosted by the US government and the military to make information easily accessible throughout different computing platforms. The first beta came to be known as ARPANET and it evolved into what is now known as the Internet or World Wide Web.

As mentioned in the previous chapter, the Internet was not designed to control hardware or do any of the sophisticated things that are now possible. The demand for remote management and real time data has driven the rise of new programming techniques that make the Web what it is today. There are many languages used for programming sites among the most popular are VBScript, JavaScript, HTML and Java. HTML is the base language for displaying websites; it is considered a static language because it writes to the browser one time only. VBScript and JavaScript are dynamic languages; meaning that they can respond to user events on a form, handle images, change colors and text on demand without the need of refreshing a web page. These are close to regular Basic, Java and C languages; functions can be written on a web page and executed depending on different user events like clicking a button or passing the mouse over an image.

In order to access a website, a web server must be running to respond to clients or users who connect to it. The server processes requests and sends back the user the

information that was called for. This introduces the concept of client and server scripting. Figure 11 represents the flow diagram between client and server via the Internet.

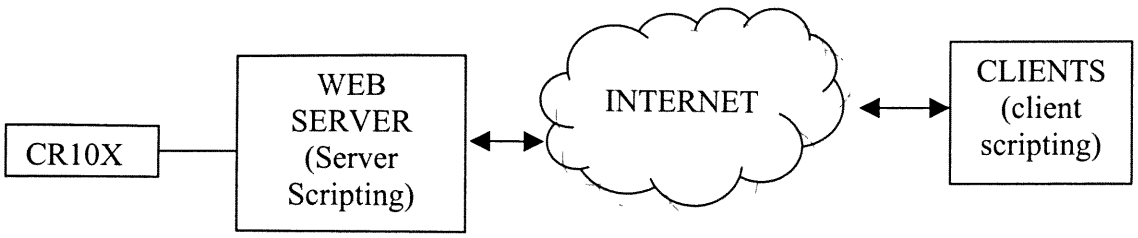


Figure 11. Representation of clients and servers connected by the Internet.

A. Client & Server Scripting

Server scripting is code that is executed at the server. Such code can be written in various languages like Active Server Pages (ASP), Java Server Pages (JSP), Pearl Helper Pages (PHP) or Common Gateway Interface (CGI). Each language is different and runs on a particular server. For example, ASP runs on Microsoft servers, JSP requires the Java scripting engine and can run in either Microsoft or Linux servers and PHP runs on Apache servers which can be mounted under Windows or Linux. The important thing about these languages is that they run on the server, so they can access databases, text files and the server's resources. When a client calls a server page, it is executed at the server and the results are sent back to the client's browser page. The web server used in this setup is a Microsoft Internet Information Services that runs ASP pages.

Client scripting runs on the web browser at the client's computer. It can be a combination of JavaScript and VBScript. They are very powerful scripting languages because they form the basis of all dynamic content on the web. This was important

because the control panel of the remote user had to emulate the one for the local main program and be able to change colors of images and change text boxes with different values. The remote GUI, also called Central Control Console (CCC) is illustrated in Figure 12.

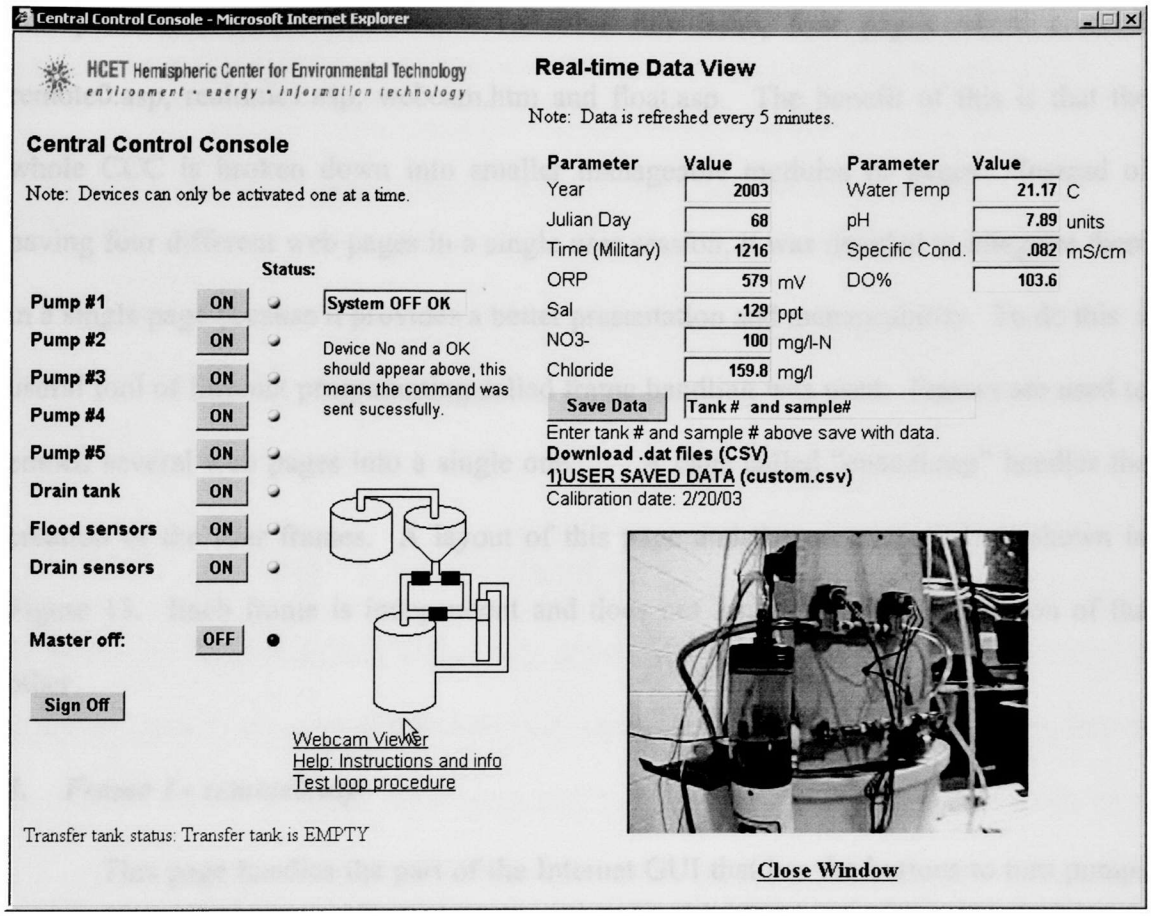


Figure 12. Central control console (CCC) for remote users, (cpanel.asp).

The CCC was kept as simple as possible and it looks like the main control panel of Figure 8. Command buttons are on the left and real time data is on the right. The CCC needs to able to handle four different tasks at the same time:

1) Handle user events when the command buttons are clicked

2) Check the status of the transfer tank constantly

3) Display real time data every 30 seconds

4) Manage the web cam viewer

All these functions could have been integrated in a single web page, but this would have caused a problem because all four events happen at different time intervals and perform different functions. To solve this issue, four pages where created: remote0.asp, realtime1.asp, webcam.htm and float.asp. The benefit of this is that the whole CCC is broken down into smaller manageable modules or pages. Instead of having four different web pages in a single user session, it was decided to integrate them in a single page because it provides a better presentation and manageability. To do this a useful tool of Internet programming called frame handling was used. Frames are used to embed several web pages into a single one. A page called "cpanel.asp" handles the creation of the four frames. A layout of this page and the program code is shown in Figure 13. Each frame is independent and does not interfere with the function of the other.

1. Frame 1 - remote0.asp

This page handles the part of the Internet GUI that has the buttons to turn pumps and valves on and off. When a button is hit, it calls a function written in JavaScript that makes a call to an ASP page to write to the text file. Every time an ASP page is called is executed at the server and the results are sent to the client. This procedure is not dynamic, the state of the previous page is lost and a new page is loaded. This was a problem because when a button is hit, a variable at the client side should retain the status of which device is active. If this page was to be reloaded, the current active device will

Frame 1 Source = "remote0.asp" Timing interval = 0 sec	Frame 2 Source="realtime1.asp" Timing interval = 30 sec
Frame 3 Source = "float.asp" Timing interval = 30 sec	Frame 4 Source = "webcam.htm" Video stream provided by yahoo.com

```
<frameset framespacing="0" border="0" cols="160,200" frameborder="0">
      <frameset rows="610,*">
            <frame name="cpanel" target="main" src=remote0.asp
scrolling="no" noresize>
            <frame name="float" src="float.asp" scrolling="no"
noresize>

      </frameset>
      <frameset rows="380,*">
            <frame name="realtimedata" src="realtime1.asp"
scrolling="no" noresize>
            <frame name="webcam" src="webcam.htm" scrolling="no"
noresize>
      </frameset>
```

Figure 13. Frameset layout and code for cpanel.asp.

be unknown. To avoid this a technology called remote scripting was used. This type of scripting consists of a set of functions that permits a JavaScript or VBScript function at the client's side to invoke a server page and wait for data without the need of reloading the page. Remote scripting is a form of remote procedure call (RPC), a term that describes the exchange of data between remote computer systems. It opens a socket connection between a remote client and the server to communicate seamlessly as if they had a full duplex link between one another. Data can be transmitted back and forth between the client and the server without ever needing to reload or rewrite the page at the client side. This is known as the User Datagram Protocol (UDP) or connectionless protocol because once the data is sent the two services are not connected to each other.

They are only connected at the instant that the client writes to the server and vice versa.

The remote scripting functions are provided by Microsoft and are designed for Internet Explorer. These tools enable the creation of an ASP page that handles the writing of commands to the text file and returning a response to the interactive page at the client side. The following lines of code shows how a user who pressed, say button 1 to turn pump 1 on, can see if this was actually written at the server.

```
1. function btnON1_onclick() {
2. cmd="000";
3. write(cmd);
4. ledoff();
5. document.status1.src=imgon;
6. document.status10.src=pumpon;}
7. function write(cmd){
8. strResults=RSExecute("Remotex.asp","writefile",cmd);
```

Button 1 points to the function in line 1, "btnON1_onclick()," so it is executed. The string "000" is assigned to the variable "cmd" and is passed to the function "write()" in line 3. Lines 4, 5 and 6 change the color of the image to red to indicate that the device is on, and to gray for the devices that are off. Line 7 shows the start of the function "write()," the RSExecute method in line 8 calls the function "writefile()" located in the ASP page "remotex.asp." At this file the following code is executed:

```
10.  function writefile(cmd)
11.  whichFN1=server.mappath("server.vtc")
12.  dim fstemp, filetemp1
13.  Set fstemp = server.CreateObject("Scripting.FileSystemObject")
14.  const forappending =8
15.  set filetemp1=fstemp.OpentextFile(whichFN1, forappending)
16.  filetemp1.writeline(cmd)
17.  filetemp1.close
```

Line 11 tells the code to use file "server.vtc", line 15 opens it for appending and line 16 writes whatever was passed by line 8, in this case "000." Finally, line 17 closes

the file. Program control is then resumed by "remote0.asp" and remains idle until another button is pressed.

Programming in this way separates the Internet GUI from the main program and permits flexibility in design. A web developer only needs to know what function codes to write (i.e. 000, 001, off, etc.), to turn a device on or off. Issues with serial communications and programming of SCADA are irrelevant to the developer. The server is in charge of reading from the text file "server.vtc" and executing all the hardware communication. Table 3 summarizes the available codes and their function.

Table 3. Available commands for web development.

Command written in "server.vtc"	Function achieved	
000	Pump #1 on	Note: only one device can be activated at one time. When a particular device is turned on, all others will be turned off!
001	Pump #2 on	
010	Pump #3 on	
011	Pump #4 on	
100	Pump #5 on	
101	Valve #1 on	
110	Valve #2 on	
111	Valve #3 on	
off	All systems off	

2. *Frame 2 – realtime1.asp*

This file handles the display of real time data. As mentioned in chapter III, the main program is constantly obtaining data from the CR10X every 10seconds. This data is displayed in the local GUI with the corresponding labeling (i.e. year, date, time, pH, etc.). Every time it is displayed, the main program writes this data to another text file called "datalog.vtc," so that the web server can have access to it. The new data string replaces the old one, so that the newest one is always available. But how to display real

time data on the web that changes dynamically? An algorithm to display data at the client side was developed, much like the one at the main program. The following piece of code from "realtime1.asp" shows the algorithm:

```
1.  function start(){
2.  var strFunction;
3.  var cmd1="document.realtime.txtPar";
4.  var cmd2=".value=strDataCSV[";
5.  var cmd3="]";
6.  //stored by server
7.  strFunction=RSExecute("remotex.asp","extractdata");
8.  strTempData=strFunction.return_value;
9.  //strData is a CSV string that must separated
10. var strDataCSV=extract(strTempData);
11. //send individual data strings to each corresponding text box
12. for (i=1;i<12;i++)
13. {
14. cmd=cmd1+i+cmd2+i+cmd3;//builds string with final instruction
15. eval(cmd);//evaluates the instruction cmd to send data to ALL
input boxes
16. }
17. timer = setTimeout("start()",30000)
18. }
```

The first issue was to have the script repeat over a period of time. When the web page loads it executes the function "start()" for the first time. When this function is finished processing at line 17, the timer function waits for 30000ms or 30sec and then it executes the function start() again. Line 7 enables remote scripting via the RSExecute method and executes the function "extractdata()" located in the file "remotex.asp":

```
19. function extractdata()
20. whichFN2=server.mappath("datalog.vtc")
21. dim fstemp2, filetemp2
22. Set fstemp2 = server.CreateObject("Scripting.FileSystemObject")
23. const forreading =1
24. set filetemp2=fstemp2.OpentextFile(whichFN2,1)
25. extractdata= filetemp2.readline
26. filetemp2.close
```

Line 20 maps the file "datalog.vtc," line 24 opens it for reading and the data that resides in the first line is stored in the variable "extractdata" in line 25. The contents of

this variable are then passed to "strDataCSV" in line 10. Lines 12-16 parse the data and extract the corresponding values as outlined in table 2 and displayed in the corresponding labels. Line 17 is reached again, and after 30 seconds the whole process is repeated. This algorithm provides real time data to the user. Active controls or java applets can be used to setup pictures or gauges that move according to the value of the data. For example a control of a thermometer can be added, if desired, to catch the temperature data and present a visual representation of the current temperature. The most important thing though, is to have all the data available to the web page. On a technical note, even though data is refreshed every 30seconds, the data of the sensors from the Hydrolab will not change until one minute has passed.

The client has the option to record a data string to a CSV file located at the server. This is done by pressing the "Save Data" button located in the CCC (please refer to Figure 12) page which calls the function "save()." Referring to the following code the sequence of steps can be followed:

```
1.  function save(){
2.  var comment, writecustom;
3.  comment=document.realtime.txtComment.value;
4.  comment=strTempData+","+comment;
5.  //call remote scripting
6.  writecustom=RSExecute("remotex.asp","writecustom",comment);
7.  comment=writecustom.return_value;
8.  comment=comment+" has been appended to 'custom.dat'!"
9.  alert(comment);
10. document.realtime.txtComment.value="";
11. document.realtime.txtComment.focus();
12. }
```

Line 4 stores the data string and any user comment on a variable conveniently called "comment." Line 6 creates the RSExecute object to enable remote scripting and call the function "writecustom()":

```
13. function writecustom(comment)
14. whichFN3=server.mappath("custom.csv")
15. dim fstemp3, filetemp3
16. ' first, create the file system object
17. Set fstemp3 = server.CreateObject("Scripting.FileSystemObject")
18. ' Now open the file
19. const forappending =8
20. set filetemp3=fstemp3.OpentextFile(whichFN3, forappending)
21. filetemp3.writeline(comment)
22. filetemp3.close
23. set filetemp3=nothing
24. set fstemp3=nothing
25. writecustom=comment
26. end function
```

Line 14 points to the file "custom.csv", line 20 opens the file for appending and line 21 writes the data to the file. Line 22 closes the file and the data is passed to the variable "writecustom" in line 25. Program control is returned in line 7 and a message box pops up indicating that the information was read successfully and stored in "custom.csv," like Figure 14 illustrates. After an experiment is over, the client can download this file and prepare the data for archiving and presentation.

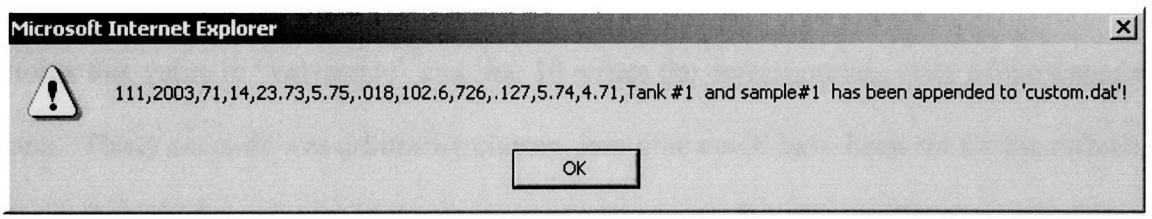


Figure 14. Confirmation of data entry.

These first two pages, "remote0.asp" and "realtime1.asp" were the ones that took more time in programming and making them work. Many codes and algorithms had to be implemented for the application of interest. One is in charge of controlling the lab setup and the other one in acquiring the data. This permits a user to run the experiment from a remote location that has a web browser and an Internet connection.

3. *Frame 3 – float.asp*

The main program also stores in a file called "levels.vtc" the status of the float level sensors. There are three states to the values of the level switches. The main program deciphers the states and writes an "EMPTY", "FULL" or "E/F" in the text file "levels.vtc." The polling method is now reversed, "float.asp" checks the text file every 30 seconds and writes the status of the transfer tank to the CCC:

```
 1.   <meta HTTP-EQUIV="refresh" CONTENT="30;url="float.asp">
 2.   <%
 3.   'Read float status file
 4.   dim FileHndl, fileobject, valvestate
 5.   Floatfile = server.mappath("levels.vtc")
 6.   set fileobject =
      server.CreateObject("Scripting.FileSystemObject")
 7.   set FileHndl = fileobject.OpentextFile(Floatfile,1)
 8.   valvestate = FileHndl.readline
 9.   FileHndl.close
10.   Response.Write "<font color=red>Transfer tank status: Transfer
      tank is " & valvestate & "</font>"
11.   %>
```

Line 1 specifies that this file should be executed at the server every 30 seconds. Line 7 opens the file "levels.vtc" and reads the status that the main program wrote. Line 8 stores this value in "valvestate" and line 10 writes the corresponding state of the transfer tank. Thirty seconds was arbitrarily chosen, less time could have been set for the refresh, depending on the urgency of the parameter of interest. For this application, this timing does not affect the overall experiment because if the transfer tank is full, the hardware stops all pumps immediately so that there is not an overflow. This method is useful for the display of control data that is required to appear in a specific interval of time.

4. *Frame 4 – webcam.htm*

This file is just an active X control that streams real time video of the remote location via a Yahoo video stream server. A video server can also be installed, if a developer does not want to use Yahoo, which is free; or this page can be omitted at all. For remote deployment a camera might not be available. That is why it is important that the status of the experiment is known. Appendix III has the full code to embed the web cam images to the HTML page.

B. Security

The last step is to implement two security measures: prevent unauthorized users from accessing the CCC and permit authorized users to access the CCC one at a time. There are many ways to accomplish security in a web server. The restricted pages can be placed inside a password-protected folder and let the web server handle security. Another possibility is to create an ASP login script that checks if the user has entered the correct login name and password, if true it continues to write the web page, otherwise it redirects the user to the login page. Both methods are rather similar and can be chosen depending on the developer. The ASP method was used in this thesis for simplicity. Figure 15 shows the typical login page that is presented to a remote user before accessing the CCC. This page is handled by "login.asp" and it follows the following code:

```
1. <%
2. 'if the form was filled out, set the session variables
3. if not Request.Form("username") = "" then
4. Session("user") = Request.Form("username")
5. Session("pass") = Request.Form("password")
6. end if
7. 'if the session variable do not match the username/password combo,
   show the log in form
8. if not (Session("user") = "srs" and Session("pass") = "srs@hcet")
   then
```

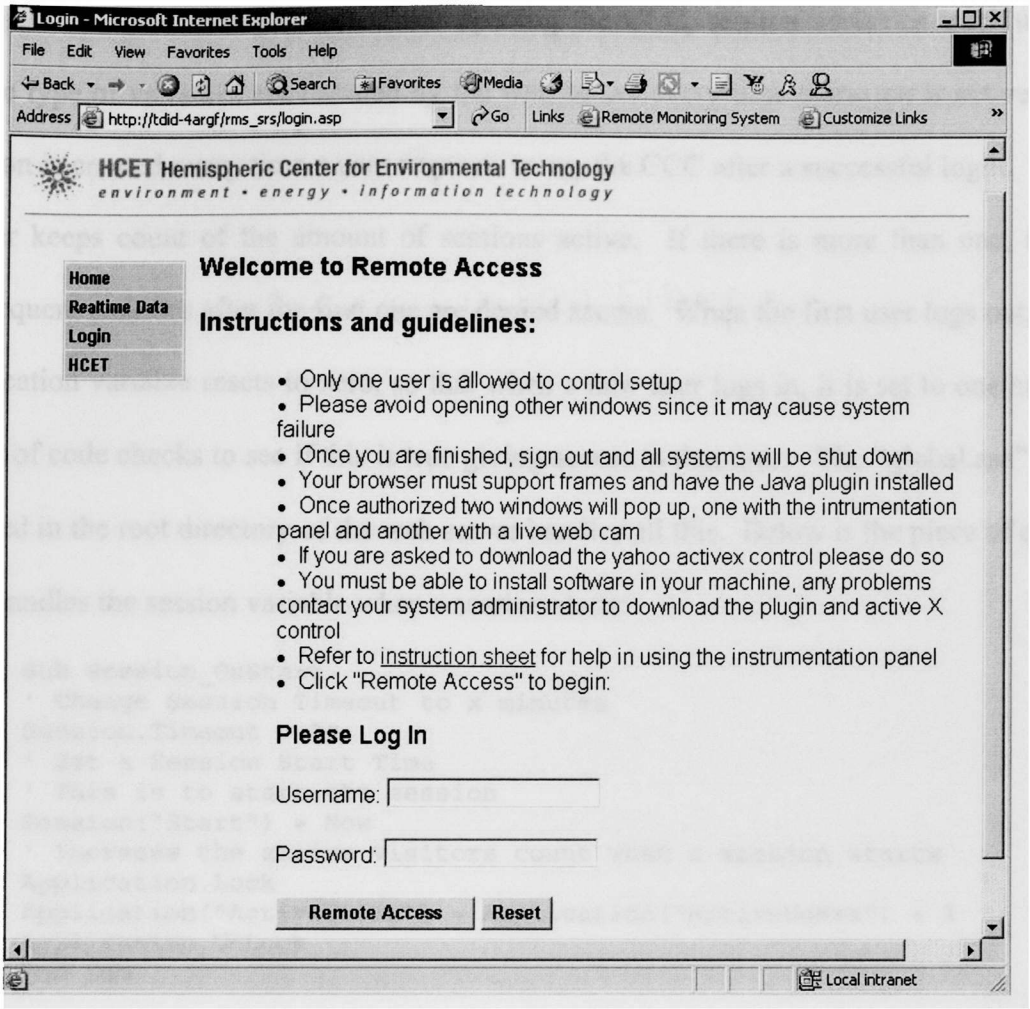


Figure 15. Login screen for a remote user

```
9.  %>
10. <LOGIN FORM HERE>
11. <% else %>
12. <!-- Log in succeeded -->
13. <SCRIPT ID=clientEventHandlersJS LANGUAGE=javascript>
14. window.open('cpanel.asp','win1','width=860,height=675');
15. </SCRIPT>
16. <% end if %>
```

Lines 3-5 extract the username and password from the fields in Figure 12. Line 8 checks to see if they are equal to the username and password noted. If the two fields match then the script proceeds to line 14 and opens the CCC; if they don't line 10 starts the login session again.

To restrict multiple users from opening the CCC, session variables were used. These type of variables are handled by the web server every time a session is active. A session is created every time a user requests to see the CCC after a successful login. The server keeps count of the amount of sessions active. If there is more than one, then subsequent sessions after the first one are denied access. When the first user logs out, the application variable resets to zero, so that when a new user logs in, it is set to one and a piece of code checks to see if this is true giving access to that user. The "global.asa" file located in the root directory of the web server handles all this. Below is the piece of code that handles the session variable when a session starts:

```
1.  Sub Session_OnStart
2.  ' Change Session Timeout to x minutes
3.  Session.Timeout = 30
4.  ' Set a Session Start Time
5.  ' This is to start the session
6.  Session("Start") = Now
7.  ' Increase the active visitors count when a session starts
8.  Application.Lock
9.  Application("ActiveUsers") = Application("ActiveUsers") + 1
10. Application.UnLock
11. End Sub
```

Line 6 starts the session and line 9 increments the amount of active sessions. There is another piece of code located at the CCC page "cpanel.asp" that checks the status of the application variable "ActiveUsers" and resets it when prompted:

```
1.  <% 'check to see if there is another user logged in, if true
    display content
2.  if     Application("ActiveUsers") = 1 then
3.  %>
4.  <CONTENT in HTML>
5.  <%else
6.  response.write("<b><FONT COLOR=#CC0000>Sorry this feature is being
    accessed by someone else, try again later!</FONT></b>")
7.  end if%>
```

Line 2 checks if the application variable is one, if is true it proceeds to line 4, this is where the code of the frames goes and is shown in appendix III. If there is more than

one user trying to access the site, line 4 is skipped and the message in line 6 is written. When a user logs out the session variable "activeusers" is reset to zero to give other clients access.

CHAPTER V

V. INTERNET PROGRAMMING FOR TCP/IP SERVER

Programming of the client interface for the TCP server is different than for the text file method. A client application can be created [3, 4, 5 and 6] that communicates with the TCP server. When creating client applications like these, they don't run inside a browser but are meant to be downloaded and installed in the client machine. If the user decides to switch computers, then the software must be installed in the new machine. A more practical approach is for the client application to be available via a web browser so that portability and accessibility are maximized. There were two web applications considered for this thesis work that can be executed in a web browser: Active X controls and Java applets. These can be programmed in different languages and compiled to execute inside a container, in this case a web browser. When the client requests these pages, the programs are downloaded in the client's computer and executed. This type of network uses the client's resources to communicate with the server, thus liberating the server of resource utilization and processing.

Client programming must follow the client-server paradigm. Before any data can be shared between the client and the server, the client must attempt to connect. It must initiate communication at the same host name: "tdid-4argf.hcet.fiu.ed" and port number:"1001," in order for the sever to accept a request. Once the server has accepted to communicate with the client, then data can be exchanged. When the client has finished, it must send a disconnect signal to the server to close the connection. At this time, the server goes back to listening mode to wait for future connection requests.

A. Active X controls

Active X are supported by Microsoft. They can be programmed in various languages like Visual Basic or Visual C++. After the control is created, it is packaged in a Internet cabinet file to be distributed to clients. When the client requests the control, all the necessary run time files and libraries are downloaded and installed at the client. These controls are mainly accessible through Windows Internet Explorer.

The adaptation of building controls from already designed projects makes them very popular. Take for example the TCP client designed for this application. Since VB was used as the main programming language, the TCP client is also programmed in VB. Figure 16 illustrates the control for the CCC as viewed by a client through a web browser. It resembles the main GUI in Figure 8 because it was derived from the same form. In other words, the GUI of the main local programmed was modified for Internet use. This process permits developers to create Active X controls to be used in the Internet quickly and efficiently.

Some modifications had to be done to the form. Since this is a TCP client, first a connect method was added to it. Second, when the buttons are clicked, they send data to the server via the Internet TCP instead of sending to the serial port. Lastly, the client must disconnect so that the sever goes back to listening mode to wait for another connection.

To connect to the server, the TCPclient is given the corresponding host and port number. The next step is to invoke the "TCPclient.connect" method and wait for the server to accept the request. When the communication link is created, data can be sent via the "TCPclient.senddata" method and data is received via the "TCPclient

dataarrival()" method. When the user has finished communicating with the server, a data string with the word "disconnect" is sent to the server, then the "TCPclient.close" is invoked to close the connection. When the server receives the word "disconnect," it closes itself and then calls the "TCPserver.listen" method to wait for future requests.

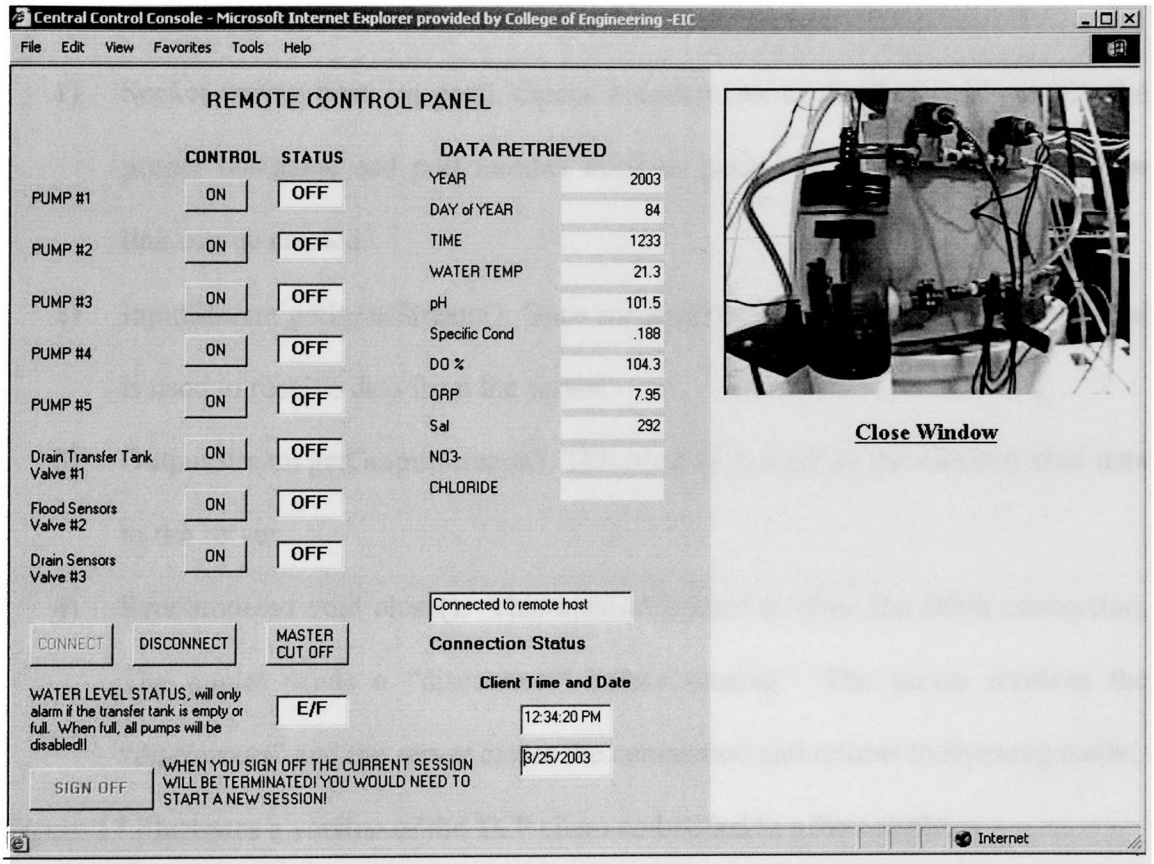


Figure 16. Active X control for CCC.

B. Java Applets

Applets are the counterpart of Active X controls. They are created with the Java programming language. Once compiled as an applet, it can be made available for access via a web browser. In order to run applets, the java runtime engine must be installed in the client's computer. Applets can be executed in various browsers instead of IE, like

Netscape and Opera. Event though they are more portable, their programming is not as straightforward as an Active X control.

The applet created for this application follows the same client-server paradigm because it is a TCP client. The difference falls in the programming language code. To initiate connection, the "java.net" [13] library of functions is used. The following methods are used to create a communication link with the server:

1) Socket (string host, int port). Opens a socket. As in the Active X control, the proper hostname and port number must be passed so that the communication link can be created.
2) InputStream getInputStream(). Once communication is established , this method is used to receive data from the server.
3) OutputStream getOutputStream(). This method is used by the client to send data to the server.
4) Synchronized void close(). This method is used to close the client connection. The applet sends a "disconnect" before closing. The server receives the "disconnect" and the server closes the connection and returns to listening mode.

Figure 17 illustrates a version of the TCP client embedded in a Java applet.

C. Security

Security is implemented the same way as in the text file client, but new methods can be applied. The pages that contain the Active X control or the Java applets can be protected by the same ASP login page used for the text file client. Another method that can be used, is to create an authentication script inside the server, so that when the client attempts to connect, a username and password would have to be provided before a

connection can be made. Both methods previously described, send command codes to the server (i.e. 000,111, off). These special codes are only understood by the server; if a code that is not recognized is received, it will be ignored. An extra layer of security can be implemented by using Secure Socket Layer connections that encrypt the data before transmission.

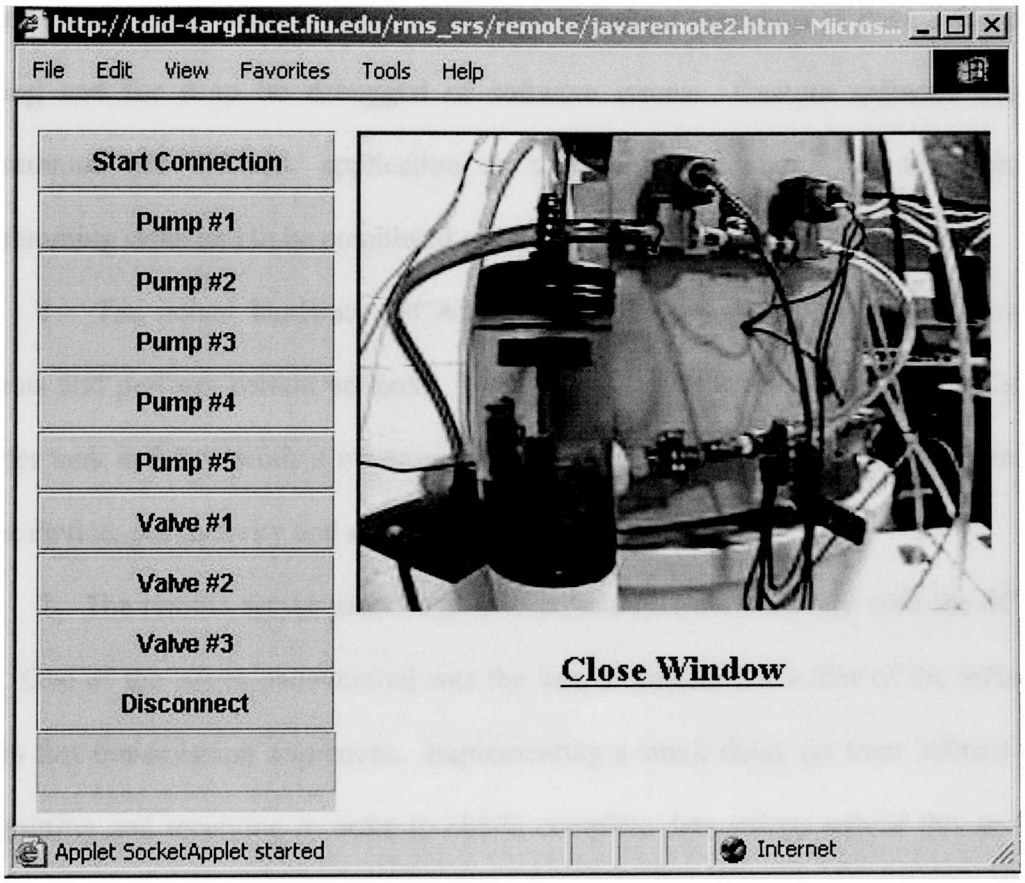


Figure 17. Java Applet for CCC.

The web site is currently hosted by HCET at the following url:

http://tdid-4argf.hcet.fiu.edu/rms_srs

CHAPTER VI

VI. RESULTS AND OBSERVATIONS

The bench test proved to be successful by letting users control the setup from a remote location without the need of being physically present. The procedure works well and can be implemented in other platforms and software to control hardware. This depends upon the developer. One of the most important aspects is to have a fast server running and for it to be debugged of software issues. Custom software must be programmed per different application as each one is unique. In this thesis, 3 programming tasks had to be completed efficiently:

1. The actual hardware (SCADA) had to be setup to "listen" to incoming requests and perform certain actions. It was also programmed to "talk" (i.e. when the transfer tank is full it sends a message indicating so). Having these small programs run on the device, productivity and speed are increased.

2. The remote server must be able to communicate effectively with the SCADA unit. One of the issues encountered was the loss of data in the buffer of the serial port due to fast transmission sequences. Implementing a small delay (at least 300ms) when transmitting and receiving in order to obtain complete data strings solved this problem. Also an algorithm that checks for valid data ensured better transmission.

3. The website interface must be dynamic and easy to use, but also foolproof. Writing to the server can be done in JSP or ASP. The client side is a combination of JavaScript, VBScript and HTML. Active X controls and Java applets are another option of contacting the server if the TCP method is used. The objective was to emulate a working GUI, as the one the main program uses to control the remote hardware.

Security was programmed by using a login script that checks and makes sure that only authorized users access the site, it should also check and permit only one session/user at a time.

The text file method was compared to the TCP server method. Users were allowed to log in to the site and control the experiment. The results of the observations were based on real time performance, stability and ease of use. Table 4 enumerates the results observed:

Table 4. Results.

	Results and Observations for client-server	
Parameter	Text file method	TCP method
Real time performance	Provides near real time performance	Provides near real time performance
Stability	Not very stable due to opening and closing of text files	Provides better stability due the TCP protocol and because the web server handles all socket communications and interrupts
Ease of use and programming	User friendly GUI can be easily programmed by using simple HTML forms, server and client scripting. Requires programming different web pages to construct one GUI.	All functions can be embedded in one Active X control or Java applet. Different programming techniques more advanced than simple client and server scripting must be used to embed this controls.

The major concern for this work was to have a stable server and to embed all remote GUI functions in a single page. Based on Table 1, the TCP server is best to employ for remote access hardware. Even though the text file method can be used as a low-end technology solution.

A. Future work and recommendations

In making the system more robust several options come to mind. Having a Microsoft server costs money and requires a decent workstation. This implementation used a 600mhz Pentium III processor with 128MB of RAM running Windows 2000 Internet Information Services. Faster processors mean faster productivity but cost more. One solution is to program the server in an embedded microcomputer that costs a few hundred dollars. Linux can be used in this system because it does not load as many resources as windows, but is powerful enough to control the hardware on board, run different applications and a web server with less computing power. Linux is also open source software and is free. This would reduce cost dramatically and make the system smaller, more portable and secure.

The remote scripting code used was produced by Microsoft and only works well with their Internet Explorer software. The setup was accessed via the latest version of Netscape and it failed to work. One solution to this problem, is to use Java remote scripting which is supported in various browsers. Bromberg [14] provides an article of different remote scripting techniques that can be cross browser compatible. The web pages also need to be coded with a different version of JavaScript that is supported by other browsers.

The actual SCADAs can be redesigned to embed all packages in a single black box. In the current setup, the CR10X communicates half duplex, when a command is sent to it, data transmission is interrupted while it servers the request, then it is resumed. This causes delay in the transmission of real time data. If the system were full duplex, it would be very stable, fast and popular.

Another issue is the opening and closing of text files by the web server and the main program. The server crashed once because a client made a request and tried to write to the text file. The server was already opened for another transaction and this caused the error. Despite this crash, the experimented trials worked. Thus the probabilities of a server crash are small but maximized when writing to it constantly in a small period of time. The best way to solve this problem is to write to memory locations at the server, this would eliminate all access problems because the operating systems control the access queue. The other problem is that current Internet technology does not have this capability. Small executables (written in C) would have to be called by a remote user to run, call functions and accomplish this. An extra programming step would have to be implemented but the advantages will make the system more stable. Future development will make this a better access method to global variables.

The TCP server is a far more stable solution than the text file. The Active X controls and Java applets must be more robust. They need to be able to handle all socket communication errors if they occur. If the server is being used, the socket connection should advice the client that someone else is using the hardware and to try again later. Since Active X controls don't work on browsers other than IE, Java applets can be setup for users that don't run IE. Or, all the development can be done in Java applets, this depends on the developer and what features each control provides.

In regards to security, a database of usernames and password should be available to give different users different levels of access, as opposed to hard coding the username and password. In this implementation, hard coding was used to simplify the design and to be able to demo the prototype.

The constraints of this thesis specify that the experiment is in house. But the final destination of the setup will be a remote site. Thus certain modifications must be done to the hardware as well as the software. Since the system will be communicating via a wireless link, the timing of transmission of the real time data will have to be increased so that the battery is not drained. Even though there is a solar panel feeding voltage to the system, this does not guarantee that there will be enough power to drive the system for the amount of time that a measurement will take place. Also, data transmission should take place only when the server has established communication with a remote client. The 110VAC pumps have to be replaced for 12-24 VDC solar pumps. This is one of the major drawbacks for deployment and future development in this area is needed.

CHAPTER VII

VII. CONCLUSION

SCADA systems are used by many industries because they can manage sensors and control external hardware. They are embedded systems that can be programmed and deployed to monitor all kinds of parameters depending on the application. As an example, the nuclear industry has SCADAs installed that check for water levels and quality at different sites.

The problem with commercially available SCADAs is that they are restricted to a local network of users and can only be accessed by proprietary software. An Internet development guide was not available to give remote users out of the network, control and access to SCADA data and external hardware through simple user interfaces. Researchers and scientists don't need to know the hardware conflicts behind these systems, they should be able to retrieve the data they need with the most minimum amount of effort. SCADAs are not usually used to control external hardware on demand, but to be deployable and autonomous systems. Their capabilities can be extended to use them to access external hardware on demand for the purpose of conducting experiments at hazardous sites via remote access.

A laboratory setup was built and interfaced to the SCADA to control a network of pumps and valves to take measurements of water quality from distinct tanks. A custom software was developed to interface the SCADA and communicate with it constantly. Once this was accomplished, a client-server paradigm methodology was applied to make this experiment available to remote users via the Internet. This consists in appending a

server module to the main program that communicates effectively with the SCADA. Two modules were used, a text file server and a TCP server.

The text server module polls a text file that is available to a web server by using server side and remote scripting. The text file is used as a global variable because all software packages have access to it. An Internet GUI was developed to give remote users access to the laboratory setup. When a web user clicks on a button in the GUI, it fires an event that writes to the text file. Since this file is available to the server module, it opens it and reads a code from it. This code is sent to the main program and passed to the original functions that send commands to the SCADA. This procedure makes the communication between the hardware and the remote user look transparent.

The TCP server module uses the Internet transport protocol to create a communication link between the client and the server. In contrast to the text file method, a remote user must first initiate communications with the server. When communication is established and a button is clicked in the GUI, data is sent via the socket connection. The data is received by the server and executed by the SCADA. The TCP server is a better method to use than the text file because it proves to be more reliable due to the fact that the web server handles all socket I/O communications. Furthermore, the TCP protocol is very efficient because it works in full duplex mode and guarantees a reliable stream of data [4].

This procedure proved to be effective because the website is independent of the main program. Some of the benefits of this study include: the creation of simple user interfaces, access to a wider research audience, and guidance for future system development of remote hardware configurations.

Many applications arise from remote management of SCADA units. These include but are not limited to accessing real time data, checking the status of the environment and controlling external hardware to do certain functions. The ease of having these setups via the Internet provides the flexibility of research from different sites and the economics of having less or no manpower in the field, not to mention protecting worker's health by minimizing exposure to possible contaminants.

LIST OF REFERENCES

[1] D. E. Comer, *Computer Networks and Internets,* 3rd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

[2] P. Arpia, A. Baccigalupi, F. Cennamo and P. Daponte, "A remote measurement laboratory for educational experiments," *Measurement*, vol. 21, pp. 157-169, Aug. 1997.

[3] Ch. Salzmann, H.A. Latchman, D. Gillet and O. D. Crisalle, "Requirements for real-time laboratory experimentation over the internet," *International Conference on Engineering Education,* ICEE'98, Rio de Janeiro, Brazil, 1998.

[4] J. W. Overstreet and A. Tzes, "An Internet-based real-time control engineering laboratory," *IEEE Control Systems Magazine,* vol. 19, issue 5, pp. 19-34, Oct. 1999.

[5] D. Gillet, H. A. Latchman, Ch. Salzmann and O. D. Crisalle, "Hands-on laboratory experiments in flexible and distance learning," *Journal of Engineering Education,* vol 90, no. 2, pp. 187-191, 2001.

[6] D. Srinivasagupta and B. Joseph, "An internet-mediated process control laboratory," *IEEE Control Systems Magazine,* vol. 23, issue 1, pp. 11-18, Feb. 2003.

[7] C. L. Churms, D. Shin, E. S. Yoon, S. J. Park and E.S. Lee, "Web-based interactive virtual laboratory system for unit operations and process systems engineering," *Computers & Chemical Engineering,* vol. 24, no. 2-7, pp. 1381-1385, 2000.

[8] P. D. Hoo, "Remote Instrumentation Access and Control," M.S. Thesis, Florida International University, Miami, FL, 1996.

[9] C. Hopp, S. Stoll and U. Konigorski, "Remote control design and implementation using the Internet" *Proceedings of the Fifth Biannual World Automation Congress (WAC 2002),* vol. 14, pp. 481-486, 2002.

[10] B. Qiu, Y. Liu, K. Chan and L. Cao, "LAN-based control for load shedding," *IEEE Computer Applications in Power,* vol. 14, issue 3, pp. 38-43, Jul. 2001.

[11] F. J. Monaco and A. Gonzaga, "Remote device command and resource sharing over the Internet: a new approach based on a distributed layered architecture," *IEEE Transactions on Computers,* vol. 51, issue 7, pp. 787-792, July 2002.

[12] "Collecting and displaying data on the web," Campbell Scientific Inc., Logan, UT, 2000.

[13] C. S. Horstmann and G. Cornell, *Core Java*, Palo Alto, CA: Sun Microsystems, Inc., 1998.

[14] P. Broomberg, "Remote Scripting with cookies for small amounts of data," [Online]. Available: http://www.eggheadcafe.com/articles/20010828.asp.

[15] M. Nelson, *Serial Communications Developer's Guide,* Foster City, CA: M&T books, 2000.

[16] B. Pfaffenberger, *Linux Networking Clearly Explained,* San Francisco, CA: Morgan Kaufmann, 2001.

[17] "CR10X Measurement and Control System Operator's Manual," Campbell Scientific Inc., Logan, UT, 2002.

APPENDICES

A. *Appendix I- Program code for embedded system*

REMOTE.DLD – CR10X ASSEMBLY PROGRAM

```
;{CR10X}
*Table 1 Program
  01: 60        Execution Interval (seconds)


1:  If Flag/Port (P91)
 1: 11          Do if Flag 1 is High
 2: 30          Then Do

2:  Batt Voltage (P10)
 1: 1           Loc [ batt       ]


3:  SDI-12 Recorder (P105)
 1: 0           SDI-12 Address
 2: 0           Start Measurement (aM0!)
 3: 8           Port
 4: 2           Loc [ WaterTemX ]
 5: 1.0         Mult
 6: 0.0         Offset

4:  If (X<=>F) (P89)
 1: 2           X Loc [ WaterTemX ]
 2: 3           >=
 3: 0           F
 4: 30          Then Do

     5:  If (X<=>F) (P89)
      1: 2          X Loc [ WaterTemX ]
      2: 4          <
      3: 50         F
      4: 30         Then Do

          6:  Block Move (P54)
           1: 1         No. of Values
           2: 2         First Source Loc [ WaterTemX ]
           3: 1         Source Step
           4: 14        First Destination Loc [ WATER_TEM ]
           5: 1         Destination Step

     7:  End (P95)

8:  End (P95)


9:  End (P95)
```

```
*Table 2 Program
  01: 10.0000   Execution Interval (seconds)


1:  Set Port(s) (P20)
 1: 8777      C8..C5 = input/output/output/output
 2: 7888      C4..C1 = output/input/input/input


2:  Read Ports (P25)
 1: 0004      Mask (0..255)
 2: 23        Loc [ LOWLVLSW  ]


3:  Read Ports (P25)
 1: 0002      Mask (0..255)
 2: 24        Loc [ HIGHLVLSW ]

4:  Do (P86)
 1: 10        Set Output Flag High (Flag 0)

5:  Set Active Storage Area (P80)^24261
 1: 1         Final Storage Area 1
 2: 111       Array ID

6:  Real Time (P77)       ^20727
 1: 1220      Year,Day,Hour/Minute (midnight = 2400)

7:  Sample (P70)      ^21468
 1: 1         Reps
 2: 14        Loc [ WATER_TEM ]

8:  Sample (P70)       ^30394
 1: 7         Reps
 2: 3         Loc [ pHX       ]

9:  Sample (P70)^5545
 1: 2         Reps
 2: 23        Loc [ LOWLVLSW  ]

10:  Serial Out (P96)
 1: 52        Printer Comma/9600 Baud

*Table 3 Subroutines

End Program

-Input Locations-
1 batt       1 1 1
2 WaterTemX  5 4 1
3 pHX        1 2 0
4 SpCX       1 2 0
5 DO_PctX    1 2 0
6 ORPX       1 2 0
7 SALX       1 2 0
```

```
8 NITRATEX   1 1 0
9 CHLORIDEX  1 2 0
10 FLAG9      1 0 0
11 __________ 1 0 0
12 __________ 1 0 0
13 __________ 1 0 0
14 WATER_TEM  1 1 1
15 PH         1 0 0
16 SPC        1 0 0
17 DO_PCT     1 0 0
18 ORP        1 0 0
19 SAL        1 0 0
20 NITRATE    1 0 0
21 CHLORIDE   1 0 0
22 __________ 0 0 0
23 LOWLVLSW   1 1 1
24 HIGHLVLSW  1 1 1
25 ENDING     1 0 0
26 __________ 1 0 0
27 __________ 1 0 0
28 __________ 1 0 0
```

B. Appendix II - Visual basic code for main program and servers

SRS1.BAS-MAIN MODULE

'AUTHOR: VICTOR ACUÑA, M.S. 2003, FLORIDA INTERNATIONAL UNIVERSITY

```
Public strComment As String
Public strFinalData As String
Public Sub send(data)

Do
   'DoEvents, wait for response of "*" from cr10x to continue
   frmCpanel.MSComm1.Output = data + Chr(13)
    'implement delay
    For i = 1 To 1000000

    Next i
   frmCpanel.MSComm1.Output = Chr(13)
   buffer$ = frmCpanel.MSComm1.Input 'Buffer$ & MSComm1.Input

Loop Until InStr(buffer$, "*")
frmCpanel.txtStatus.text = frmCpanel.txtStatus.text + buffer$ + vbCr

End Sub
Public Sub turnAction(strCmd1 As String)
If strCmd1 = "" Then
Exit Sub
End If

turnEnable (0) 'activate decoder
   Select Case strCmd1
   'this switches C6,C5,C4 into a binary 3 bit number for the decoder
switching
        Case "000" 'select decoder and turn on corresponding sequence
            send ("9104:0:0370U") 'syntax 91xx:y:ffffU = xx port #, y=
0 for low, 1 for high, ffff is checksum
            send ("9105:0:0371U") 'syntax 9104 refers to port 4, 0 make
low, checksum
            send ("9106:0:0372U") 'checksum is sum of all ASCII
characters up to the colon
        Case "001"
            send ("9106:0:0372U")
            send ("9105:0:0371U")
            send ("9104:1:0371U")

        Case "010"
            send ("9104:0:0370U")
            send ("9105:1:0372U")
            send ("9106:0:0372U")


        Case "011"
            send ("9106:0:0372U")
```

```
            send ("9105:1:0372U")
            send ("9104:1:0371U")
        Case "100"
            send ("9106:1:0373U")
            send ("9105:0:0371U")
            send ("9104:0:0370U")
        Case "101"
            send ("9106:1:0373U")
            send ("9105:0:0371U")
            send ("9104:1:0371U")
        Case "110"
            send ("9106:1:0373U")
            send ("9105:1:0372U")
            send ("9104:0:0370U")
        Case "111"
            send ("9106:1:0373U")
            send ("9105:1:0372U")
            send ("9104:1:0371U")
        Case "off"
            send ("9104:0:0370U")
            send ("9105:0:0371U")
            send ("9106:0:0372U")
            Exit Sub
        Case Else
            'do nothing, ignore
    End Select
turnEnable (1)
End Sub

Public Sub turnEnable(intCmd2 As Integer)
send (Chr(13)) 'data logger needs some CR to wake up and wait for
commands

If intCmd2 = 0 Then 'shutdown enable line and reset all ports to zero
    send ("9107:0:0373U") 'sets enable low, turns driver ckt off

End If
If intCmd2 = 1 Then
    send ("9107:1:0374U") 'sets enable high, turn driver ckt on

End If
End Sub
```

FRMCPANEL CODE FOR SRS1.FRM

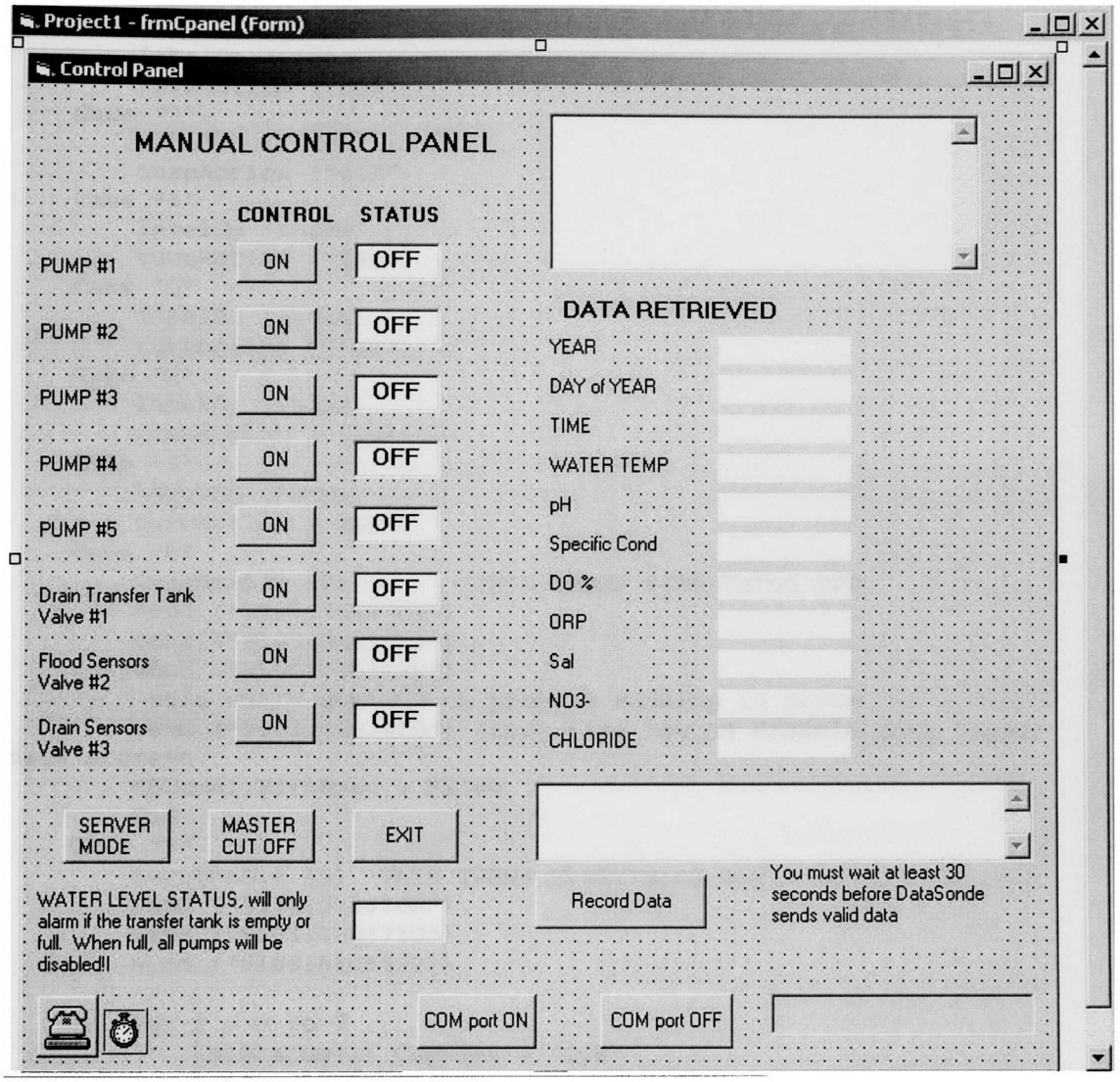


Figure 18. frmCpanel.

```
Dim headcount As Integer
Dim indat As String
Dim data As String

Private Sub cmdButton_Click(Index As Integer)
Dim i As Integer

Select Case Index

    Case "0"
        labelOn (Index)  'turn label on
        turnAction ("000") 'send decoder address
```

```
        Case "1"
            labelOn (Index)
            turnAction ("001")
        Case "2"
            labelOn (Index)
            turnAction ("010")
        Case "3"
            labelOn (Index)
            turnAction ("011")
        Case "4"
            labelOn (Index)
            turnAction ("100")
        Case "5"
            labelOn (Index)
            turnAction ("101")
        Case "6"
            labelOn (Index)
            turnAction ("110")
        Case "7"
            labelOn (Index)
            turnAction ("111")
        Case "8"
            turnEnable (0)        'THIS TURNS EVERYTHING OFF
            send ("9104:0:0370U")
            send ("9105:0:0371U")
            send ("9106:0:0372U")
            'this entry depends on program running in cr10x
            send ("9001:0:0366U") 'turn flag low to disable data logger
data storage
            MSComm1.PortOpen = False
            End
        Case "9"
            turnEnable (0) 'THIS TURNS EVERYTHING OFF
            send ("9104:0:0370U")
            send ("9105:0:0371U")
            send ("9106:0:0372U")

            For i = 0 To 7
                lblStatus(i).Caption = "OFF"
                lblStatus(i).BackColor = &H80000009
            Next i

        End Select

    End Sub

    Private Sub labelOn(intStatusOn As Integer)
        For i = 0 To 7
            lblStatus(i).Caption = "OFF"
            lblStatus(i).BackColor = &H80000009
        Next i

        lblStatus(intStatusOn).Caption = "ON"
        lblStatus(intStatusOn).BackColor = &HFF&
    End Sub
```

```
Private Sub cmdOff_Click()
frmCpanel.MSComm1.PortOpen = False
lblCom.Caption = "COM Port is OFF"
For i = 0 To 9
cmdButton(i).Enabled = False
Next i
End Sub

Private Sub cmdOn_Click()
frmCpanel.MSComm1.PortOpen = True
lblCom.Caption = "COM Port is ON"
For i = 0 To 9
cmdButton(i).Enabled = True
Next
End Sub


Private Sub cmdServer_Click()
response = MsgBox("Entering server mode will disable local control" & _
    " and give access to remote users only.  Do you wish to continue?
", vbInformation + vbYesNo, "Server_Mode")

If response = vbYes Then

    turnEnable (0) 'THIS TURNS EVERYTHING OFF
    send ("9104:0:0370U")
    send ("9105:0:0371U")
    send ("9106:0:0372U")
    Me.Hide
    frmServer.Show
    frmServer.Timer1.Enabled = True
End If

End Sub

Private Sub Form_Initialize()


frmCpanel.MSComm1.PortOpen = True
If frmCpanel.MSComm1.CommEvent = comPortAlreadyOpen Then
    MsgBox ("Com port is in use by another program!")
End

End If

turnEnable (0)
lblCom.Caption = "COM Port is ON"
'turn flag 1 high to initiate program in data logger, this entry
'depends on the program running in the data logger
send ("9001:1:0367U")
headcount = 0
indat = ""
End Sub
```

```
Private Sub Form_Terminate()
  frmCpanel.MSComm1.PortOpen = True
  turnEnable (0)
  send ("9104:0:0370U")
  send ("9105:0:0371U")
  send ("9106:0:0372U")
  'this entry depends on program running in cr10x
  send ("9001:0:0366U") 'turn flag low to disable data logger data
storage
  frmCpanel.MSComm1.PortOpen = False

End Sub

Private Sub Levels(levelsw As String)
levelsw = Left(levelsw, 2)

If levelsw = "40" Then
    lblLevel(1).Caption = "EMPTY"
    lblLevel(1).BackColor = &HFF&
ElseIf levelsw = "02" Then
    lblLevel(1).Caption = "FULL"
    lblLevel(1).BackColor = &HFF&
ElseIf levelsw = "00" Then
    lblLevel(1).Caption = "E/F"
    lblLevel(1).BackColor = &H80000009
End If

Open "e:/downloads/rms_srs/rms_srs_local/levels.vtc" For Output As #2
Print #2, lblLevel(1).Caption
Close #2

End Sub

Private Sub MSComm1_OnComm()
Dim strDataRaw As String

    'store incoming data
    strDataRaw = frmCpanel.MSComm1.Input

    If headcount = 3 Then
        If strDataRaw = Chr$(10) Then
            'split comma separated values to get individual values
            strTempdata = Split(indat, ",")
            If UBound(strTempdata) = 13 Then
                headcount = 696
            Else
                indat = ""
                headcount = 0
            End If
        End If
        indat = indat & strDataRaw
    End If
```

```
    If strDataRaw <> "1" And headcount > 0 And headcount < 3 Then
        headcount = 0
    End If

    If strDataRaw = "1" And headcount < 3 Then
        headcount = headcount + 1
    End If

    'debugging windows
    txtData.text = txtData.text & strDataRaw

    'data must be filtered before writing


    If headcount = 696 Then
        'if resulting string is not the whole data set, discard
        'strTempdata(0) = Right(strTempdata(0), 3) 'to separate
location id from illegal characters

        For i = 1 To 11 '11
            lblDataSet(i - 1) = strTempdata(i) & " "
        Next i

        'rebuild data nicely for output
        strFinalData = Join(strTempdata, ",")

        'send float level switches status

        Levels (strTempdata(12) & strTempdata(13))

        headcount = 0
        indat = ""

        Open "e:/downloads/rms_srs/rms_srs_local/datalog.vtc" For
Output As #1
        Print #1, strFinalData
        Close #1
    End If


End Sub

Private Sub cmdRecord_Click()
'append new data to data file
     Dialog.Show

End Sub


Private Sub Timer1_Timer()
    send (Chr(13))
End Sub
```

DIALOG CODE FOR DIALOG.FRM

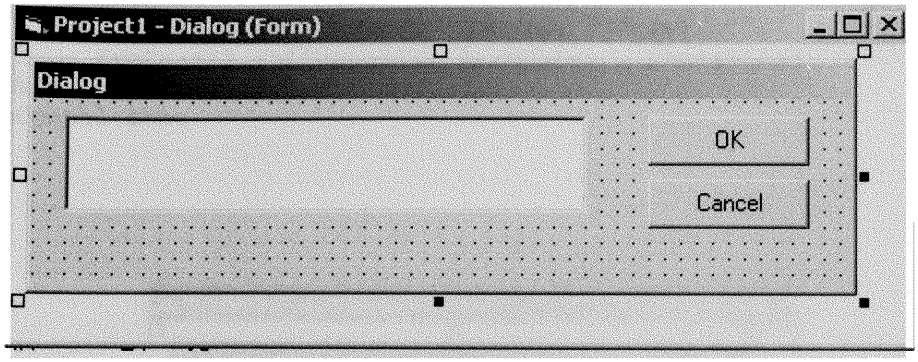


Figure 19. frmDialog.

```
Dim strComment As String
Option Explicit

Private Sub CancelButton_Click()
Me.Hide
End Sub


Private Sub OKButton_Click()
strComment = txtComment.text
strFinalData = strFinalData & "," & strComment

     Open "e:/downloads/cr10x/logger.dat" For Append As #1
     Print #1, strFinalData
     Close #1
     MsgBox ("Record has been added")
'reset textbox
txtComment.text = vbNullString
Me.Hide
End Sub
```

FRMSERVER CODE FOR SERVER.FRM (TEXT FILE SERVER)

```
Private Sub Command1_Click()
lstStatus.Clear
End Sub

Private Sub cmdLocal_Click()
response = MsgBox("This will disable server mode and return control" &
_" to local users only.  Do you wish to continue? ", _
    vbInformation + vbYesNo, "Local_Mode") If response = vbYes Then
    turnEnable (0) 'THIS TURNS EVERYTHING OFF
    send ("9104:0:0370U")
    send ("9105:0:0371U")
    send ("9106:0:0372U")
```

```
    Me.Hide
    frmServer.Timer1.Enabled = False
    For i = 0 To 7
            frmCpanel.lblStatus(i).Caption = "OFF"
            frmCpanel.lblStatus(i).BackColor = &H80000009
    Next i
    frmCpanel.Show
End If
End Sub
```

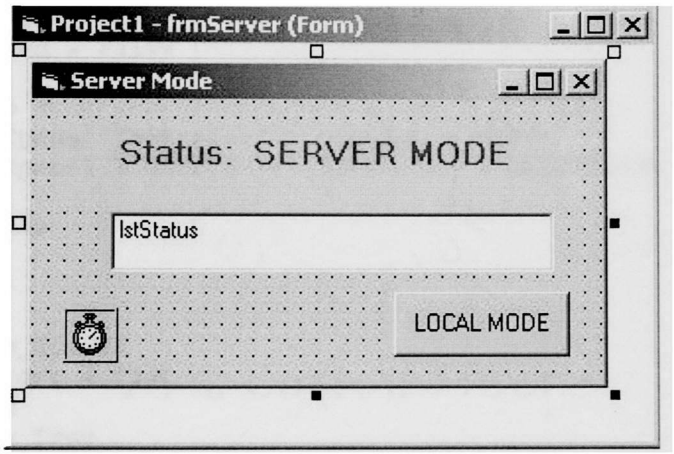


Figure 20. frmServer.

```
Private Sub Timer1_Timer()

Open "e:/downloads/rms_srs/rms_srs_local/server.vtc" For Input As #1

Dim text As String

    Do Until EOF(1)
        Line Input #1, text
            If text = "" And EOF(1) = True Then
                Close #1
                Exit Sub
            Else
                lstStatus.AddItem text
                turnAction (text)
            End If
    Loop

Close #1

Open "e:/downloads/rms_srs/rms_srs_local/server.vtc" For Output As #1
Print #1, vbNullString
Close #1

End Sub
```

FRMTCPSERVER CODE FOR TCPSERVER.FRM (TCP SERVER)

```
Private Sub cmdLocal_Click()
response = MsgBox("This will disable server mode and return control" &
_" to local users only.  Do you wish to continue? ", _
    vbInformation + vbYesNo, "Local_Mode")

If response = vbYes Then
    turnEnable (0) 'THIS TURNS EVERYTHING OFF
    send ("9104:0:0370U")
    send ("9105:0:0371U")
    send ("9106:0:0372U")
    tcpServer.Close
    Timer1.Enabled = False
    Me.Hide
    For i = 0 To 7
            frmCpanel.lblStatus(i).Caption = "OFF"
            frmCpanel.lblStatus(i).BackColor = &H80000009
    Next i
    frmCpanel.Show
End If
End Sub

Private Sub Form_Load()
tcpServer.LocalPort = 1001
tcpServer.Listen
Timer1.Enabled = True
Call TCPstate
End Sub

Private Sub Form_Terminate()
tcpServer.Close
frmCpanel.Show
Timer1.Enabled = False
Me.Hide
End Sub


Private Sub tcpServer_ConnectionRequest(ByVal requestID As Long)

    If tcpServer.State <> sckClosed Then _
    tcpServer.Close
    ' Accept the request with the requestID
    ' parameter.
    tcpServer.Accept requestID

End Sub

Private Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
Dim TCPdata As String
tcpServer.GetData TCPdata
If TCPdata = "disconnect" Then
        tcpServer.Close
        tcpServer.Listen
        turnAction ("off")
```

```
Else
turnAction (TCPdata)
End If
Call TCPstate
End Sub

Private Sub tcpServer_SendComplete()
txtDatasent.text = strFinalData
Call TCPstate
End Sub

Private Sub Timer1_Timer()
If tcpServer.State = sckConnected Then
    tcpServer.SendData strFinalData
End If
Call TCPstate
End Sub

Private Sub TCPstate()
Label1.Caption = tcpServer.State
End Sub
```

C. *Appendix III - Internet code for remote users*

GLOBAL.ASA – MANAGE SESSIONS

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>

Sub Application_OnStart
      ' Set user count to 0 when the server starts
      Application("ActiveUsers") = 0
End Sub

Sub Session_OnStart
      ' Change Session Timeout to x minutes
      Session.Timeout = 30
      ' Set a Session Start Time
      ' This starts the session
      Session("Start") = Now
      ' Increase the active visitors count
      Application.Lock
            Application("ActiveUsers") = Application("ActiveUsers") + 1
      Application.UnLock
End Sub

Sub Session_OnEnd

End Sub

</SCRIPT>
```

LOGIN.ASP – AUTHENTICATES USERS

```
<%@ Language=VBScript %>
<HTML>
<HEAD><title>Login</title>
</HEAD>
<BODY><!--#include file="menu.htm"-->
<H3>Welcome to Remote Access</H3>
<H3>Instructions and guidelines:</H3>



    <TABLE style="WIDTH: 600px" cellSpacing  =1 cellPadding=1
    width=600 align=left border=0 background="">

      <TR>
        <TD style="WIDTH: 50px" width=50></TD>
        <TD style="WIDTH: 500px" width=500>

  <LI>Only one user is allowed to control setup

  <LI>Please avoid opening other windows since it may
      cause system failure
```

```
  <LI>Once you are finished, sign out and all systems
      will be shut down
      <LI>  Your browser must support frames and have the Java plug in
installed
      <LI>Once authorized two windows will pop up, one with the
instrumentation panel and another with a live web cam
      <LI>If you are asked to download the yahoo activex control please
do so
      <LI>You must be able to install software in your machine, any
problems
      contact your system administrator to download the plug-in
and active X
      control 
      <LI>  Refer to <A class=download href="help.html" target=top >
instruction sheet</A> for help
            in using the instrumentation panel

  <LI>Click "Remote Access" to begin:  <!--
login.asp ***
 This is the page which you will be redirected to if your log in failed
or was timed out.
-->

<%
'if the form was filled out, set the session variables
if not Request.Form("username") = "" then
 Session("user") = Request.Form("username")
 Session("pass") = Request.Form("password")
end if

'if the session variable do not match the username/password combo, show
the log in form
if not (Session("user") = "srs" and Session("pass") = "srs@hcet") then
%>

<form name="form1" method="post" autocomplete="off">
  <p align="left"><font size="4">Please Log In</font></p>
  <p>Username: <input name="username"
     ></p>
  <p>Password: <input type="password" name="password"></p>
  <p>
    <input type="submit" name="Submit" value="Remote Access">
    <input type="reset" name="Reset" value="Reset">
  </p>
</form>

<% else %>

 <!-- Log in succeeded -->
<SCRIPT ID=clientEventHandlersJS LANGUAGE=javascript>
window.open('cpanel.asp','win1','width=900,height=690');
</SCRIPT>

 <p align="center"><font size="5">Thank You</font></p>
```

```
<p>
 Thanks for logging in. You are authorized to access private
 sections of this web site.
</p>

<% end if %></LI>

      </TD></TR></TABLE>

</BODY>
</HTML>
```

CHECKLOGIN.ASP – CHECKS TO MAKE SURE VISITOR IS STILL LOGGED IN

```
<!--
' checklogin.asp ***
' This file checks to see if you have logged in. If you have not, it
will redirect you to the login page.
' If you have logged in, it will continue to load the rest of the
requested page.
-->

<%
'This will keep the requested page in the address bar.
if not (Session("user") = "srs" and Session("pass") = "srs@hcet") then
 Server.Transfer("login.asp")
end if
%>
```

CPANEL.ASP – CALLS THE CONTROL PANEL AND REAL TIME DATA VIEW

```
<% 'check to see if there is another user logged in
if    Application("ActiveUsers") = 1 then
%>

<!-- #include file="checklogin.asp"-->
<HTML>
<HEAD>
<title>Central Control Console</title>
</head>


<frameset framespacing="0" border="0" cols="160,200" frameborder="0">
      <frameset rows="610,*">
            <frame name="cpanel" target="main" src=remote0.asp
scrolling="no" noresize>
            <frame name="float" src="float.asp" scrolling="no"
noresize>

      </frameset>
      <frameset rows="380,*">
            <frame name="realtimedata" src="realtime1.asp"
scrolling="no" noresize>
```

```
            <frame name="webcam" src="webcam.htm" scrolling="no"
noresize>
      </frameset>
    <noframes>
  <body>

  <p>This page uses frames, but your browser doesn't support them.</p>
  <p>This page uses frames, but your browser doesn't support them.</p>

  </body>
   </noframes>
</frameset>

</html>
<%
 Session("user") = ""
 Session("pass") = ""
%>
<%else
      response.write("<b><FONT COLOR=#CC0000>Sorry this feature is
being accessed by someone else, try again later!</FONT></b>")
      'Session.Abandon
end if%>
```

REMOTE0.ASP – CONTROL PANEL INTERFACE AND FUNCTIONS

```
<html><head>

<title>Central Control Console</title>
<base target="main">

</head>
<body onload="start()" onunload="return window_onunload()">

<script language="JavaScript" src="rs/rs.htm"></script>
<script language="JavaScript">RSEnableRemoteScripting("rs/");</script>
<!--#include file="style1.css"-->
<img height="40" src="images/logo2.gif" width="350"><br><br>
<h3><font color=blue>Central Control Console</h3></font>Note: 
Devices can only be
activated one at a time.<br>

<form name="form1"> 
<table style="WIDTH: 400px" cellSpacing="2" cellPadding="2" width="400"
align="left" border="0" background="">

  <tr>
    <td style="WIDTH: 175px" width="175"></td>
    <td style="WIDTH: 50px" width="50"></td>
    <td style="WIDTH: 40px" width="40" colSpan="2">
      <p align="center"><strong><font face="Arial"
size="2">Status:</font></strong></p></td>
```

```
    <td style="WIDTH: 30px" width="30">
      <p align="center"> </p></td></tr>
  <tr>
    <td><strong>Pump #1</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON1_onclick()" type="button" size="28" value="ON" name="btnON1"
LANGUAGE="javascript"></td>
    <td style="WIDTH: 10px" width="10">
      <p align="left"><IMG height=14 src="images/whitedot.gif" width=14
align=left name=status1></p></td>
    <td style="WIDTH: 10px" width="10"></td>
    <td><input id="txtStatus1" style="WIDTH: 116px; HEIGHT: 22px"
size="13" name="txtStatus1" readOnly></td></tr>
  <tr>
    <td><strong>Pump #2</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON2_onclick()" type="button" size="28" value="ON"
name="btnON2"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status2></td>
    <td></td>
    <td rowSpan="3">
      <p><font face="Arial" size="2">Device No and a OK <br>should
appear above, this
      </font><font face="Arial" size="2">means the command was
sent
      sucessfully. </font></p>
  </td></tr>
  <tr>
    <td><strong>Pump #3</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON3_onclick()" type="button" size="28" value="ON"
name="btnON3"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status3></td>
    <td></td></tr>
  <tr>
    <td><strong>Pump #4</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON4_onclick()" type="button" size="28" value="ON"
name="btnON4"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status4></td>
    <td></td></tr>
  <tr>
    <td><strong>Pump #5</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON5_onclick()" type="button" size="28" value="ON"
name="btnON5"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status5></td>
    <td></td>
    <td></td></tr>
  <tr>
    <td><strong>Drain tank</strong></td>
```

```
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON6_onclick()" type="button" size="28" value="ON"
name="btnON6"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status6></td>
    <td></td>
    <td rowSpan="6" valign="center"><IMG height=184
src="images/system_off.gif" width=160 name=status10></td></tr>
  <tr>
    <td><strong>Flood sensors</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON7_onclick()" type="button" size="28" value="ON"
name="btnON7"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status7></td>
    <td></td></tr>
  <tr>
    <td><strong>Drain sensors</strong></td>
    <td><input style="WIDTH: 42px; HEIGHT: 24px" onclick="return
btnON8_onclick()" type="button" size="28" value="ON"
name="btnON8"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status8></td>
    <td></td></tr>
  <tr>
    <td style="HEIGHT: 25px" height="25"></td>
    <td></td>
    <td></td>
    <td></td></tr>
  <tr>
    <td><strong>Master off:</strong> </td>
    <td><input language="javascript" id="button2" onclick="return
btnOff_onclick()" type="button" value="OFF" name="btnOff"></td>
    <td><IMG height=14 src="images/whitedot.gif" width=14 align=left
name=status9></td>
    <td></td></tr>
  <TR>
    <TD style="HEIGHT: 50px" vAlign=bottom height=50><INPUT
language=javascript style="FONT-WEIGHT: bold" onclick="return
btnSignOff_onclick()" type=button value="Sign Off"
name=btnSignOff></TD>
    <TD></TD>
    <TD></TD>
    <TD></TD></TR>
  <tr>
    <td style="HEIGHT: 50px" vAlign="bottom" height="50"></td>
    <td>

</td>
    <td></td>
    <td colSpan=2>
      <P><A class=download href="webcam.htm" target=top>Webcam
Viewer</A><br>
      <A class=download href="help.html" target=top >Help: Instructions
and info</A>
```

```
      <br><A class=download href="procedures.htm" target=top >Test loop
procedure</A></P></td></tr></table>
<p>           &n
bsp;           &
nbsp;         
</p>
<p></p></form></body></html>
</form>

<script ID="clientEventHandlersJS" LANGUAGE="javascript">

var cmd
var imgon="images/reddot.gif", imgoff="images/whitedot.gif";
var pumpon="images/pump_on.gif", systemoff="images/system_off.gif",
valve1="images/valve_1.gif";
var valve2="images/valve_2.gif", valve3="images/valve_3.gif"; //image
sources
var serverURL="remotex.asp";
var strResults,strStatus;  //string to hold results from remote server

function start()
{
btnOff_onclick();
}

function write(cmd)
{
//this is used to call an asp page and pass a javascript variable
//document.location.replace('remote0.asp?Var=' + cmd);

//Call RSExecute to enable remote scripting and return the values to
the array
//remember syntax variable=RSExecute("page_name.asp","function_name");
//then convert strings to float using Parse float and
//finally update the input fields and use round to restrict 3 decimal
places

strResults=RSExecute(serverURL,"writefile",cmd);
strStatus=strResults.return_value; //store complete string

//select proper label for status check
switch (strStatus) {
case "000":
      strStatus="Pump #1 OK";
      break
case "001":
      strStatus="Pump #2 OK";
      break

case "010":
      strStatus="Pump #3 OK";
      break
case "011":
      strStatus="Pump #4 OK";
      break
```

```
case "100":
      strStatus="Pump #5 OK";
      break
case "101":
      strStatus="Valve #1 OK";
      break
case "110":
      strStatus="Valve #2 OK";
      break
case "111":
      strStatus="Valve #3 OK";
      break
case "off":
      strStatus="System OFF OK";
      break
default:
      strStatus="";
      break
}

form1.txtStatus1.value=strStatus;
//turn all leds off
}

function ledoff(){

//image variables
var cmd1="document.status"; cmd2=".src=imgoff";

//turn all leds off
for (i=1;i<10;i++)
      {eval(cmd1+i+cmd2);
}
}

function btnON1_onclick() {
//cmd=form1.txtInput1.value;
cmd="000";
write(cmd);
ledoff();
document.status1.src=imgon;
document.status10.src=pumpon;
}

function btnON2_onclick() {
cmd="001";
write(cmd);
ledoff();
document.status2.src=imgon;
document.status10.src=pumpon;
}

function btnON3_onclick() {
cmd="010";
write(cmd);
```

```
ledoff();
document.status3.src=imgon;
document.status10.src=pumpon;
}

function btnON4_onclick() {
cmd="011";
write(cmd);
ledoff();
document.status4.src=imgon;
document.status10.src=pumpon;
}

function btnON5_onclick() {
cmd="100";
write(cmd);
ledoff();
document.status5.src=imgon;
document.status10.src=pumpon;
}
function btnON6_onclick() {
cmd="101";
write(cmd);
ledoff();
document.status6.src=imgon;
document.status10.src=valve1;
}
function btnON7_onclick() {
cmd="110";
write(cmd);
ledoff();
document.status7.src=imgon;
document.status10.src=valve2;
}
function btnON8_onclick() {
cmd="111";
write(cmd);
ledoff();
document.status8.src=imgon;
document.status10.src=valve3;
}


function btnOff_onclick() {
cmd="off";
//document.location.replace('remote0.asp?Var=' + cmd);
write(cmd);
ledoff();
document.status9.src=imgon;
document.status10.src=systemoff;
}


function btnSignOff_onclick() {
cmd="off";
```

```
write(cmd);
ledoff();
document.location.replace('reset.asp');
top.window.close();
}

function window_onunload() {
cmd="off";
write(cmd);
document.location.replace('reset.asp')
}

</script>
```

RESET.ASP – RESETS APPLICATION VARIABLE

```
<%
Application("ActiveUsers")=0
%>
```

REALTIME1.ASP – EXTRACTS DATA AND DISPLAYS IT ON SCREEN

```
<HTML>
<HEAD>
<title>Real-time Data View</title>
</HEAD>

<script language="JavaScript" src="rs/rs.htm"></script>
<script language="JavaScript">RSEnableRemoteScripting("rs/");</script>

<script language="javascript">
strDataCSV = new Array;
var serverURL="remotex.asp";
var strTempData; //catches string of data

function start(){
var strFunction;
var cmd1="document.realtime.txtPar";
var cmd2=".value=strDataCSV[";
var cmd3="]";

//RSExecute calls extract data on server and gets the latest value
stored by server
strFunction=RSExecute(serverURL,"extractdata");
strTempData=strFunction.return_value;

//strData is a CSV string that must separated
var strDataCSV=extract(strTempData);

//send individual data strings to each corresponding text box
for (i=1;i<12;i++)
      {
```

```
      cmd=cmd1+i+cmd2+i+cmd3;//builds string with final instruction
      eval(cmd);//evaluates the instruction cmd to send data to ALL
input boxes
      //document.realtime.txtPar1.value=strDataCSV[0];
      }
timer = setTimeout("start()",30000)
}
//function to calculate size of incoming array of data
function findSize(temp){
var count=0;
for (n=1; n<=temp.length; n++)
      {
            if (temp.substring(n-1,n)==",")
            {
            count++;//count amount of commas
            //remember actual size of final array will be count+1
            }
      }
return count;
}


//function to extract comma separated data
function extract(temp)
{
var size=findSize(temp);
//size gives array size (size+1) this adapts to any string
for(m=0; m<size+1; m++)
{
      for (n=1; n<=temp.length; n++)
      {
            if (temp.substring(n-1,n)==",")
            {
            //catch first element before ","
            strDataCSV[m]=temp.substring(0,n-1);
            //rewrite the original string without the first value
            temp=temp.substring(n,temp.length);
            n=100;//cheap way to break out of for loop so that values
don't get mixed up
            }
      }
}
//catch last accumulated value
strDataCSV[size]=temp;
return strDataCSV;//send to function calling
//you now have an array made of the csv string passed
}

//this function writes to 'custom.dat'

function save(){
var comment, writecustom;
comment=document.realtime.txtComment.value;
comment=strTempData+","+comment;
//call remote scripting
```

```
writecustom=RSExecute(serverURL,"writecustom",comment);
comment=writecustom.return_value;
comment=comment+" has been appended to 'custom.dat'!"
alert(comment);
document.realtime.txtComment.value="";
document.realtime.txtComment.focus();
}

</script>
<!--#include file="style1.css"-->

<BODY onload="start()">
<strong><font face="Arial" color="blue" size="4">
Real-time Data View</font></strong>
<P>Note:  Data is refreshed every 5 minutes.</P>
<P>
<form name=realtime>
<TABLE style="WIDTH: 400px" cellSpacing=2 cellPadding=2 width=400
background=""
border=0 align=left>

  <TR>
    <TD style="WIDTH: 50px" width=50></TD>
    <TD style="WIDTH: 110px" width=110><STRONG>Parameter</STRONG></TD>
    <TD style="WIDTH: 50px" width=50>
      <P align=right><STRONG>Value</STRONG></P></TD>
    <TD style="WIDTH: 20px" width=20></TD></TR>
  <TR>
    <TD></TD>
    <TD>Year</TD>
    <TD><INPUT size=6 align=right name=txtPar1
      border=1
     style="TEXT-ALIGN: right" readOnly
     ></TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>Julian Day </TD>
    <TD><INPUT

      align=right size=6  name=txtPar2 style="TEXT-ALIGN: right"
readOnly
     ></TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>Time (Military)</TD>
    <TD><INPUT

      align=right size=6  name=txtPar3 style="TEXT-ALIGN: right"
readOnly
     ></TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
```

```
    <TD>Water Temp</TD>
    <TD><INPUT

      align=right size=6  name=txtPar4 style="TEXT-ALIGN: right"
readOnly
     > C</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>pH</TD>
    <TD><INPUT

      align=right size=6  name=txtPar5 style="TEXT-ALIGN: right"
readOnly
     > unit</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>Specific Cond.</TD>
    <TD><INPUT

      align=right size=6  name=txtPar6 style="TEXT-ALIGN: right"
readOnly
     > mS/cm</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>DO%</TD>
    <TD><INPUT

      align=right size=6  name=txtPar7 style="TEXT-ALIGN: right"
readOnly
     > Sat</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>ORP</TD>
    <TD><INPUT

      align=right size=6  name=txtPar8 style="TEXT-ALIGN: right"
readOnly
     > mV</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>Sal</TD>
    <TD><INPUT

      align=right size=6  name=txtPar9 style="TEXT-ALIGN: right"
readOnly
     > ppt</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>NO3-</TD>
```

```
    <TD><INPUT

      align=right size=6  name=txtPar10 style="TEXT-ALIGN: right"
readOnly
     > mg/l-N</TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD>Chloride</TD>
    <TD><INPUT style="TEXT-ALIGN: right" readOnly align=right size=6
      name=txtPar11> mg/l</TD>
    <TD></TD></TR>
  <TR>
    <TD style="HEIGHT: 5px" height=5></TD>
    <TD></TD>
    <TD></TD>
    <TD></TD></TR>
  <TR>
    <TD></TD>
    <TD><INPUT type=button value="Save Data" name=btnSave
onclick="return save()"></TD>
    <TD colSpan=2><INPUT id=text1
      style="WIDTH: 234px; HEIGHT: 22px" size=28 name=txtComment
     ></TD></TR>
  <TR>
    <TD></TD>
    <TD colSpan=3>You may enter a comment in the text
      field above to be appended to this data record.</TD></TR>
  <TR>
    <TD></TD>
    <TD colSpan=3><STRONG>Download
      .dat files (CSV)<br> right click, select "save target
      as"<BR></STRONG><A class=download href="datalog.csv"
target=top><STRONG>1)LOGGED
      DATA</STRONG></A><STRONG> (datalog.csv)<BR></STRONG><A
class=download
      href="custom.csv" target=top><STRONG>2)USER SAVED
DATA</STRONG></A><STRONG>
      (custom.csv)</STRONG></TD></TR>
  <TR>
    <TD></TD>
    <TD></TD>
    <TD> Calibration date: 2/20/03</TD>
    <TD></TD></TR></TABLE></form></P>
<P> </P>

</BODY>
</HTML>
```

FLOAT.ASP – POLLS TEXT FILE TO FIND STATE OF FLOAT SWICTHES

```
<html>
<head>
      <meta HTTP-EQUIV="refresh" CONTENT="30;url="float.asp">
```

```
</head>
<!--#include file="style1.css"-->
<body>
<%
'Read float status file
dim FileHndl, fileobject, valvestate
Floatfile = server.mappath("levels.vtc")

set fileobject = server.CreateObject("Scripting.FileSystemObject")

set FileHndl = fileobject.OpentextFile(Floatfile,1)
valvestate = FileHndl.readline
FileHndl.close


      Response.Write "<font color=red>Transfer tank status: Transfer
tank is " & valvestate & "</font>"
%>
</body>
</html>
```

REMOTEX.ASP – REMOTE SCRIPTING SERVER FILE

```
<!--<%@ LANGUAGE=VBSCRIPT %>-->
<% RSDispatch %>
<!--#INCLUDE FILE="rs/rs.asp"-->

<SCRIPT RUNAT=SERVER Language=javascript>
function Description()
{

//this.add = Function( 'n1','n2','return addNumbers(n1,n2)' );
this.writefile = Function('cmd','return writefile(cmd)');
this.extractdata = Function('return extractdata()');
this.writecustom = Function('comment','return writecustom(comment)');
}
public_description = new Description();

</script>

<SCRIPT RUNAT=SERVER LANGUAGE="VBScript">

'---------------------------------------------------------------------

function writefile(cmd)
whichFN1=server.mappath("server.vtc")

dim fstemp, filetemp1

' first, create the file system object
Set fstemp = server.CreateObject("Scripting.FileSystemObject")

' Now open it
const forappending =8
```

```
set filetemp1=fstemp.OpentextFile(whichFN1, forappending)
filetemp1.writeline(cmd)
filetemp1.close

set filetemp1=nothing
set fstemp=nothing

writefile=cmd
end function

'--------------------------------------------------------------------

function extractdata()

whichFN2=server.mappath("datalog.vtc")
dim fstemp2, filetemp2

' first, create the file system object
Set fstemp2 = server.CreateObject("Scripting.FileSystemObject")
const forreading =1

set filetemp2=fstemp2.OpentextFile(whichFN2,1)
extractdata= filetemp2.readline
filetemp2.close

set filetemp1=nothing
set fstemp=nothing
end function

'--------------------------------------------------------------------

function writecustom(comment)
whichFN3=server.mappath("custom.csv")

dim fstemp3, filetemp3

' first, create the file system object
Set fstemp3 = server.CreateObject("Scripting.FileSystemObject")

' Now open the file
const forappending =8

set filetemp3=fstemp3.OpentextFile(whichFN3, forappending)
filetemp3.writeline(comment)
filetemp3.close

set filetemp3=nothing
set fstemp3=nothing

writecustom=comment
end function

'--------------------------------------------------------------
</script>
```

WEBCAM.HTM – EMBEDS VIDEO STREAM IN A WEB PAGE

```
<html>
<head>
<title>Remote Video Cam</title>
<script>
window.onerror=function(){return true}

function s() {
VwrDlg.style.display="";
tmp.style.display="none";
}

</script>
</head>
<body onload="s()" bgcolor="aliceblue">

<div align="center">
<object
codebase="http://chat.yahoo.com/cab/yvwrctl.cab#version=1,0,0,16"
id="VwrDlg" classid="CLSID:E504EE6E-47C6-11D5-B8AB-00D0B78F3D48"
style="display:none" width="320" height="240" ></object>
</div>
<script>
VwrDlg.CountryCode="us"
VwrDlg.AppName=3
VwrDlg.Age=20
VwrDlg.RoomName=""
VwrDlg.UserName=""
VwrDlg.TargetName="d087project"
VwrDlg.Token=""
VwrDlg.Start();
</script>
<center><h3><a href="javascript:window.close()">Close
Window</a><h3></center>
</body>
</html>
```

JAVA APPLET CODE (SOCKETAPPLET.JAVA)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.util.*;
import java.io.*;
import java.net.*;

public class SocketApplet2 extends JApplet implements Runnable {

      private JButton buttons[];
      private String names[] = {"Start Connection", "Pump #1",
      "Pump#2", "Pump #3", "Pump #4", "Pump #5","Valve #1", "Valve #2",
      "Valve #3", "Disconnect" };
      private JTextField textField = null;
```

```
private GridLayout layout;
private Container container;
private String webServerStr = null;
private Socket connection = null;
private PrintWriter out = null;
private BufferedReader in = null;
private Thread thread;

public void init(){
      layout = new GridLayout(11,1);
      container = getContentPane();
      container.setLayout(layout);

      ButtonHandler handler = new ButtonHandler();

      buttons = new JButton[names.length];

      for (int i = 0; i < buttons.length; i++){
            buttons[i] = new JButton(names[i]);
            container.add(buttons[i]);
            buttons[i].addActionListener(handler);
            buttons[i].setSize(20,10);
      }

      textField = new JTextField(25);
      textField.setEditable(false);
      textField.setBackground(Color.white);
      container.add(textField);

}

private class ButtonHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
            if (e.getSource() == buttons[0])
                  startConnection();
            else if (e.getSource() == buttons[1])
                  sendBytes("000");
            else if (e.getSource() == buttons[2])
                  sendBytes("001");
            else if (e.getSource() == buttons[3])
                  sendBytes("010");
            else if (e.getSource() == buttons[4])
                  sendBytes("011");
            else if (e.getSource() == buttons[5])
                  sendBytes("100");
            else if (e.getSource() == buttons[6])
                  sendBytes("101");
            else if (e.getSource() == buttons[7])
                  sendBytes("110");
            else if (e.getSource() == buttons[8])
                  sendBytes("111");
            else if (e.getSource() == buttons[9])
          closeConnection();
      }
}
```

```
        public void startConnection(){
          try {
              connection = new Socket("tdid-4argf.hcet.fiu.edu", 1001);
                    out = new PrintWriter(connection.getOutputStream(),
true);
              in = new BufferedReader(new InputStreamReader(

        connection.getInputStream()));
            } catch (UnknownHostException e) {
              System.err.println("Don't know about host.");
              System.exit(1);
          } catch (IOException e) {
              System.err.println("Couldn't get I/O for "
                                 + "the connection to: host");
              System.exit(1);
          }
              thread= new Thread(this);
              thread.start();
        }

        public void sendBytes(String bytes){
              out.print(bytes);
              out.flush();
        }

        public void getResponseFromServlet(){

          try{
              textField.setText(in.readLine());
          }
          catch (IOException e){
              System.out.println("There was an error reading from the
servlet.");
          }
        }

        public void closeConnection(){

              out.print("disconnect");
              out.flush();

              try {
                    out.close();
                    in.close();
                    connection.close();
                    thread.destroy();
              }
              catch (IOException ioe){
                    System.out.println("Could not close connection!!");
                    ioe.printStackTrace();
              }
        }

        public void run(){
              System.out.println("hello1");
```

```
      while(true){
       try{
            //String s=in.readLine();
            //if (s!=null && !s.equals("")){
            System.out.println("hello2"+in.readLine());
            textField.setText(in.readLine());
            //}
        }
        catch (IOException e){
            System.out.println("There was an error reading from the
servlet.");
        }

      System.out.println("hello3");
      }

      }

}
```

ACTIVE X CONTROL CODE (REMOTECLIENT.DOB)

```
Dim data As String
Dim status As String

Private Sub cmdButton_Click(Index As Integer)
Dim i As Integer
 If tcpClient.State = sckClosed Then
    MsgBox ("Connection is closed, try connecting again!")
    Exit Sub
End If

Select Case Index

    Case "0"
        SendData ("000")
        labelOn (0)
    Case "1"
        SendData ("001")
        labelOn (1)
    Case "2"
        SendData ("010")
        labelOn (2)
    Case "3"
        SendData ("011")
        labelOn (3)
    Case "4"
        SendData ("100")
        labelOn (4)
    Case "5"
        SendData ("101")
        labelOn (5)
    Case "6"
        SendData ("110")
        labelOn (6)
    Case "7"
```

```
        SendData ("111")
        labelOn (7)
    Case "8"
        SendData ("off")
         For i = 0 To 7
            lblStatus(i).Caption = "OFF"
            lblStatus(i).BackColor = &H80000009
        Next i
    Case "9"
        SendData ("off")
        For i = 0 To 7
            lblStatus(i).Caption = "OFF"
            lblStatus(i).BackColor = &H80000009
        Next i
    End Select

End Sub

Private Sub labelOn(intStatusOn As Integer)
    For i = 0 To 7
        lblStatus(i).Caption = "OFF"
        lblStatus(i).BackColor = &H80000009
    Next i

    lblStatus(intStatusOn).Caption = "ON"
    lblStatus(intStatusOn).BackColor = &HFF&

End Sub



Private Sub Levels(levelsw As String)

levelsw = Left(levelsw, 2)

If levelsw = "40" Then
    lblLevel(1).Caption = "EMPTY"
    lblLevel(1).BackColor = &HFF&
ElseIf levelsw = "02" Then
    lblLevel(1).Caption = "FULL"
    lblLevel(1).BackColor = &HFF&
ElseIf levelsw = "00" Then
    lblLevel(1).Caption = "E/F"
    lblLevel(1).BackColor = &H80000009
End If

End Sub
Private Sub SendData(cmd As String)
    If tcpClient.State = sckClosed Then
        MsgBox ("Connection is closed, try connecting again!")
    Else
    tcpClient.SendData cmd
    End If
    status = cmd
End Sub
```

```
Private Sub cmdConnect_Click()
    tcpClient.Close
    tcpClient.Connect
    cmdConnect.Enabled = False
    cmdDisconnect.Enabled = True
    cmdSignoff.Enabled = False
    For i = 0 To 8
            cmdButton(i).Enabled = True
    Next i
End Sub

Private Sub cmdDisconnect_Click()
    For i = 0 To 7
            lblStatus(i).Caption = "OFF"
            lblStatus(i).BackColor = &H80000009
    Next i
    SendData ("disconnect")
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    cmdSignoff.Enabled = True
    For i = 0 To 8
            cmdButton(i).Enabled = False
    Next i
End Sub

Private Sub cmdSignoff_Click()
    For i = 0 To 7
        lblStatus(i).Caption = "OFF"
        lblStatus(i).BackColor = &H80000009
    Next i
    SendData ("disconnect")
    cmdConnect.Enabled = False
    cmdDisconnect.Enabled = False
    cmdSignoff.Enabled = False
    For i = 0 To 8
            cmdButton(i).Enabled = False
    Next i
End Sub


Private Sub tcpClient_Connect()
txtStatus.Text = "Connected to remote host"
End Sub

Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
'Dim strDataRaw As String, strDataRaw2 As String
tcpClient.GetData strDataRaw, vbString
strDataRaw = Split(strDataRaw, ",")


        If UBound(strDataRaw) = 11 Then
        For i = 1 To 9 '11
            lblDataSet(i - 1) = strDataRaw(i) & " "
        Next i
```

```
        Levels (strDataRaw(10) & strDataRaw(11))
        End If


End Sub

Private Sub tcpClient_SendComplete()
txtStatus.Text = status & " Command Sent successfully"
If status = "disconnect" Then
    tcpClient.Close

End If
End Sub

Private Sub Timer1_Timer()
currentTime = Time
currentDate = Date

Text1.Text = currentTime
Text2.Text = currentDate

End Sub
```