

Design and Implementation of Data Transfer Software

By

Candise Wise

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

March 2001

Design and Implementation of Data Transfer Software

By

Candise Wise

Submitted to the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements
for
the Degree of Bachelor Science
in Information Engineering Technology.

Copyright 2000 Candise P. Wise

The author grants to the Information Engineering Program permission to
Reproduce and distribute copies of this document in a whole or in part.

Candise Wise

Author: Candise P. Wise

3-15-01

Date:

Russell E McMahon

Faculty Project Advisor

3/4/01

Date:

Lawrence B. Sullivan

Department Head

3/7/01

Date:

Acknowledgement

I wish to express sincere gratitude to the faculty of the Information Engineering Technology program, without whose help the successful completion of this project would not have been possible. I would also like to make a special acknowledgement of the contribution made by Professor McMahon, my project advisor. I thank him for his guidance, support, and for giving freely of his time.

This senior design project would not have been possible without the unselfish help of my friend Robert Brown. I would also like to acknowledge my friends Pam Naber, Robin Miller, Caryle Clear, Ashika Wijeskera, Jason Stroh and Mark Cummings for their moral support and encouragement throughout this project.

I would also like to thank Signalysis, Inc for giving me the chance to design this project and demonstrate my abilities as a programmer. I thank them for all their help and guidance throughout this project.

Table of Contents

	Page
Acknowledgements	i
Table of Contents	ii
List of Figures	iii
Abstract	iv
1. Statement of the Problem	1
1.1 Definition of the Need	2
2. Review of the Research	2
2.1 Winspc	3
2.2 Class Function	4
2.3 ActiveX Control/ActiveX Exe	4
2.4 Winsock using TCP Protocol	5
2.5 Winsock using UDP protocol	6
3. Description of the Solution	6
3.1 User Profile	8
3.2 Design Protocols	8
4. Objectives of the Project	11
5. Design and Development	12
5.1 Budget	12
5.4 Hardware	12
5.3 Software	12
5.4 Timeline	13
6. Proof of Design	15
6.1 ActiveX Control Functionality	15
6.2 ActiveX Exe Functionality	24
7. Conclusion and Recommendations	47
8. Notes	50
9. References	53

List of Figures

Figure 1. Current production line setup of Trico	2
Figure 2. WinSpc displaying Statistical Process Data (SPC) in graphical format	3
Figure 3. Outline of new production line	8
Figure 4. Connection through SigQC and WinSpc through TCP/IP connection	10
Figure 5. Breakdown of Project Development for Fall 2000	13
Figure 6. Breakdown of Project Development for Fall 2001	14
Figure 7. SigQC Test Application and ActiveX Control	17
Figure 8. ActiveX Exe receiving information from ActiveX Control	26

Abstract

This report covers the perceived problem, solution, research, design objectives, conclusions, and recommendations of my senior design project. The purpose of the project was to solve a production line problem for Trico, a factory in Mexico, which tests windshield wiper motors. An interface with the current production line monitoring system was needed along with reliable data transfer and statistical process packet capabilities.

The data transfer project is a software application, which takes calibration-measured data on a production line, transfers the data by TCP/IP over a network into statistical process packets. The statistical process packets are then sent and broken down into graphical format and displayed in an application called Winspc. This is all done through an ActiveX control, located on the production line and ActiveX exe, located on a production manager station.

Design and Implementation of Data Transfer Software

1. Statement of Problem

A software application named SigQC, designed by Signalysis has been installed at a company called Trico. Trico specifically specializes in testing windshield wiper motors. SigQC is designed for manufacturers to automate a process for accepting or rejecting units on an assembly line based on measured noise and vibration data (7, pg. 51). The program itself works by vibration and acoustic data acquisition control, time and frequency analysis, pass/fail criteria, narrowband analysis, and percent octave analysis. Pareto charts and full reporting capabilities display the information. For Trico, SigQC was installed on seven production lines throughout the company. Each station (Figure 1, pg. 2) consists of a conveyor belt along with a separate application of SigQC testing the production line. Each system contains a separate database. Each test stand has a minimum of four acceptance tests running every 15 seconds. A production manager must monitor and maintain each production lines various failure modes.

The problem is:

- Each production line must be shut down 15-30 minutes to transfer data to the network.
- For one production manager this is very time consuming,
- For every minute the production line is shut down this is very costly to Trico.
- Trico would like data to be broken down once it has been measured and placed into Statistical Process Packets (SPC)

Plant Floor Diagram

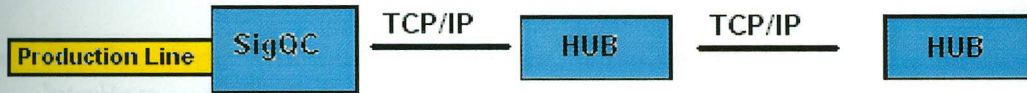


Figure 1. Current production line setup of Trico

1.1 Definition of Need

Because Signalysis designed SigQC for Trico, it was up to us to adjust the program to best fit customer specifications. Due to the problems they are experiencing in their factory along with their desire to have data transferred in to Statistical Process Packets, we need to design a completely new product. This product must not only incorporate their needs but also allow us to have a cross platform application (11, pg.50), which Signalysis can use along with any of their software applications. Since Statistical Process Packets are used frequently in production line testing, it would benefit Signalysis to have a product that could be offered along with SigQC at the time of purchase.

2. Research

The first step to solving this problem was to research various Statistical Process Packet programs (SPC) in order to have broken down data from SigQC transferred into graphical format across a network. Various companies' software applications were researched and studied in order to find out which program best fit Trico's needs and had the functionality to interface with SigQC.

The SPC software that best fit the need was a program called WinSpC (13, pg. 52) designed by Datanet. The reasons for choosing (Figure 2, pg. 4):

- WinSpC is thoroughly tested to run on Windows 95/98/NT and Windows 2000.
- WinSpC works with multiple databases, including Oracle, MS SQL Server, InterBase and Sybase.
- WinSpC harnesses the full power of multi-threading to show multiple events (14, pg.52) on the same computer. For example, you can view a document or scroll backwards on a chart, even while collecting data.
- Ability to use a single PC to simultaneously collect data from multiple parts, processes or machines. Onscreen manual data collection can occur simultaneously with background automated data collection, saving you both space and money by reducing the required number of workstations.
- Active Interface, which allows software developers to design compatible applications, which automatically send data to WinSpC from separate software applications.

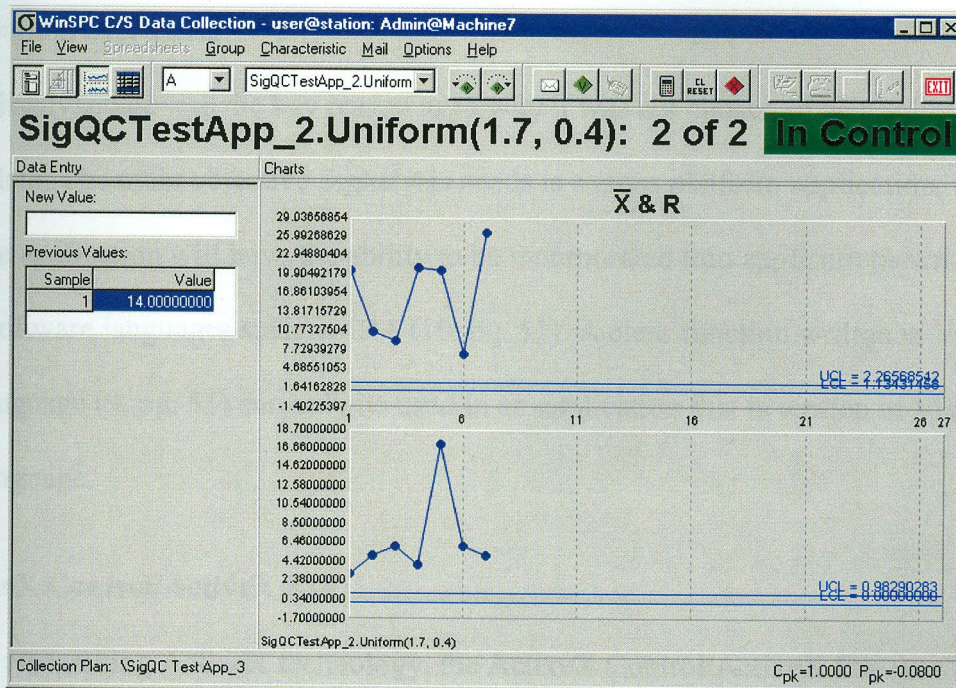


Figure 2. WinSpC displaying Statistical Process Data (SPC) in graphical format

Once WinSpc was chosen, research was done on four possible solutions for transferring data from SigQC into WinSpc. The research was done by getting information from books and websites (1-8, pg. 52). The four possible solutions that were researched were:

- Class Function
- ActiveX Control/ActiveX Exe
- WinSock using TCP Networking
- WinSock using UDP (8, pg.51). Networking

2.2 Class Function

Although the class function has many benefits, such as:

- Faster speed within program (because of less code)
- Do not have to register in Windows registry
- Built in executable

The disadvantage was the key reason as to why it was not chosen to be a solution to this problem. The main objective Signalysis needs is a cross platform application, this means the application will have the ability to be incorporated into applications written in various software languages such as C++ (10, pg. 51). A class function written in Visual Basic Language (9, pg. 51) can only be used in an application that is written in Visual Basic Language.

2.3 ActiveX Control/ActiveX Exe

After researching ActiveX technology, the ActiveX Control/ActiveX Exe was found to be a:

- Cross Platform Application
- Inherent basic client (1, pg. 50) /server+ structure.

Although the ActiveX Control does have three slight disadvantages:

- Must install ActiveX control and ActiveX Exe separately.
- Must register in Windows registry.
- Slower speed performance.

The key feature to using ActiveX technology is that it is a cross platform application.

It gives the ability for two objects to interact and communicate with each other over a network of systems regardless of their platform and language.

2.4 WinSock using TCP Protocol

For the networking area of the project, TCP protocol has many benefits for this particular project.

- There must be a connection between Server (1, pg.50) and Client
- No limit to size of data
- Guarantees delivery of data

The disadvantage to using TCP protocol with this application is extra code must be written in the application to detect the server, which reduces speed. The advantages heavily weighed out the disadvantage. The guarantee of messages is the most important factor for the Networking side of this project. In order for the production line manager to accurately monitor pass/fail criteria, the data must be received. In the case of TCP if data is not received there is an error message sent allowing for the system to be fixed. If there is no connection, an error message will be returned. Different testing procedures require

various amounts of data to be transferred; this requires no limit on the amount of data transmitted.

2.5 WinSock using UDP Protocol

After researching UDP protocol networking. The disadvantages helped to make the decision to use TCP protocol. Although UDP protocol would have:

- Faster speed
- Less Code

The disadvantages did not meet the requirements to solve the problem. By using UDP protocol:

- There is no guarantee of the order of messages
- Maximum size of data is limited
- There is no guarantee data will be received

Although the speed of UDP is faster because it is written with less code, UDP will allow for data to be sent and not received. There is no error message if data is not transmitted. Instead the data is simply never received. Unlike TCP, there does not have to be a connection to the server to send data. The order of the data sent might not be transmitted in the order it is sent. For this project, the messages need to be accurate and in order for the production manager to analyze the pass/fail criteria statistics.

3. Description of Solution

After researching each possible solution and comparing the advantages and disadvantages of each possible solution, the following were chosen as the best way to solve the problem:

- Visual Basic Programming Language will be less time consuming than C++, making it more cost efficient.
- ActiveX Control/ActiveX Exe will slow down the program but only by milliseconds, the SigQC Rate is at 30-40 second intervals, the decrease will not cause interference with the program. ActiveX is a cross platform application and will be a benefit to past and future Signalysis software applications.
- WinSock using TCP Protocol through TCP/IP Network is a solid-state connection meaning reliable data transfer. Data cannot be sent unless there is a solid connection between the Server/Client. If data is not transferred and received an error message will notify the user.

Various stations will be set up throughout Trico for the purpose of receiving data through ActiveX capabilities. This ActiveX control will package data from each production line, and transfer it LAN. Specific computers will be set up with An ActiveX exe designed to unpackage the data and in a graphical format display the information on another computer. (Figure 2, pg. 4)

Production lines will no longer have to shut down in order for data to be transferred; instead the test procedures can be monitored as they are running. The ActiveX exe (Server) will be linked through TCP/IP making it possible for the production manager to view any production lines information. It will provide an open interface toolkit in order to expand into statistical process control such as WINSPC packages.

(Figure 3, pg. 9)

Enhanced Plant Floor Diagram

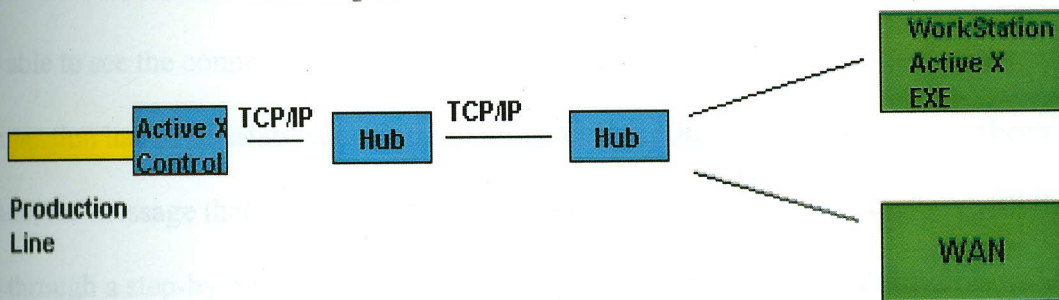


Figure 3. Outline of New Production line

3.1 User Profile

The targeted users of this system are Trico engineers. Trico is a company located in Mexico that builds windshield wiper motors. Each motor is tested on a production line measuring pass/fail limits. All engineers have gone through various computer system training and have basic computer skills. All specifications and functions of this system have been designed to their specific needs and abilities.

The industrial standards for software development are basic: the form must be gray, all text black and small buttons. Colors are not to be used on the form itself. Pictures that represent an object, such as the graphs or production line may have color but they must be basic colors. No bright colors can be used, such as neon.

3.2 Design Protocols

ActiveX Control

The ActiveX Control will be used with SigQC. When running a production line the option to export will be given. Once the option is selected, a small form will be

displayed. A textbox is the main item on this form. Once the option is selected, the ActiveX control will send a message making the TCP/IP connection. The user will then be able to see the connection state through the textbox. This allows the user to see that the connection has been established without any problem. Once the connection has been made, the message that opens WinSpc will be automatically sent. The textbox will then go through a step-by-step display of the actions that WinSpc is going through in order to load accurately. If the connection is lost, a message will appear saying the connection is lost.

The ActiveX control is linked to a SigQC program. The ActiveX control is stored on the same computer system running the production line. As the production line tests various windshield wiper motors, the measured data will be sent to the ActiveX control. Once data begins to be sent, a message box appears saying that random variables are being sent. Communication is achieved between the ActiveX Control and ActiveX exe through TCP/IP networking. As variables are taken from data measurement on the production line, they are sent to the ActiveX exe. The variables are then sent to the control located on the production line station (See Figure 4, pg 10).

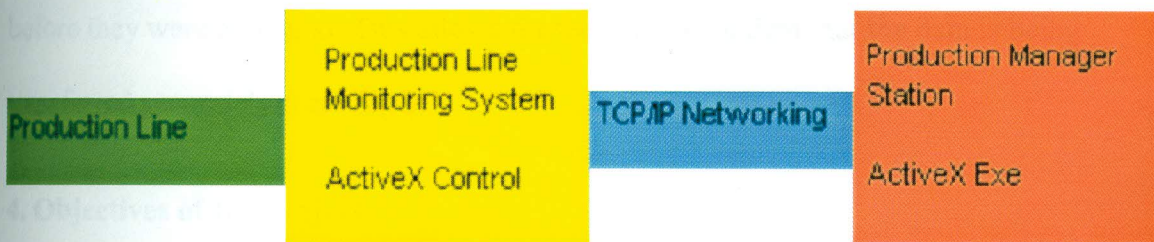


Figure 4. Connection of SigQC and WinSpc through TCP/IP connection

ActixeX Exe

The ActixeX exe will be located on production manager stations. This will give the production line manager the ability of watching the pass/fail criteria as the data is being generated. Once the ActixeX exe receives the message to open WinSpc, it will go through various stages. Because WinSpc is a detailed program it must first create folders, parts, subdirectories, sample sizes, limits and graphical settings. The ActixeX exe tells each of these various functions to open and load. A password is needed to log into each function. The user will not have to enter any type of information during the loading of WinSpc. Because it has all been coded within the ActixeX exe, there will be no type of information needed by the user to access. Once all the information is loaded WinSpc will be displayed and visible to the user.

Once data starts being received by the control it will add the data to the WinSpc Data Collection. A message will be displayed that data is being received. As the variables are being received WinSpc will average the variables, and display them in a graphical format. Functionality allows for graphical formats to be changed while data is being transferred. Spreadsheets can also be selected to display the variables that have been sent before they were averaged. This allows for the user to confirm that the data is being transferred accurately from SigQC.

4. Objectives of the project

In order to find an accurate and reliable solution, four main objectives were used to compare each possible solution. Each solution was researched and advantages and disadvantages were compared to conclude which possible solution was best for this particular need. The comparison was based on the following five main objectives:

- The program must be cost effective.
- The application needs to provide effective instant access performance without interference to SigQC application speed.
- Cross Platform Application
- Reliable Data Transfer
- The SPC software application must be compatible with SigQC.

I have consulted with Signalysis software engineers on this project and the conclusion has been made that the best way to solve this problem is to design an ActiveX Control/ActiveX Exe. ActiveX technology is the state of the art technology due to the fact it is a cross platform application. It allows the user to store the control in the toolbox of Visual Basic for future use in applications. Inherent basic client /server structure allows the control to be easily used in applications requiring client/server transmission throughout a Network. An ActiveX control exposes properties, methods and events to developers.

The objective to designing an ActiveX Control written in Visual Basic Language is to provide a cost efficient means to solving Trico's problem. The ActiveX Control will provide effective instant access performance without interference with the SigQC applications performance. After the application has been designed and completed:

1. Trico will have ActiveX Control /ActiveX Exe application with added export feature, lowering quality control costs and incorporating reliable data transference.
2. Signalysis will have cross platform application to incorporate in Signalysis applications.
3. WinSpc will incorporate all the Statistical Process Packet functionality needed by Trico.

The solution to the problem covers all the key objectives and research shows it to be the most reliable means to solving the problem.

5. Design and Development

5.1 Budget

Total Cost: \$56,000

The software and hardware resources required for the completion of this project are available at Signalysis Corporation.

5.2 Hardware

B and K Pulse System - \$40,000

5.3 Software

SigQC software - \$8,000

Visual Basic 6 - \$1,800

WinSpc Software - \$7, 000

5.4 Timeline

The following timelines show the process that was followed in order to design the data transference software in the most efficient way:

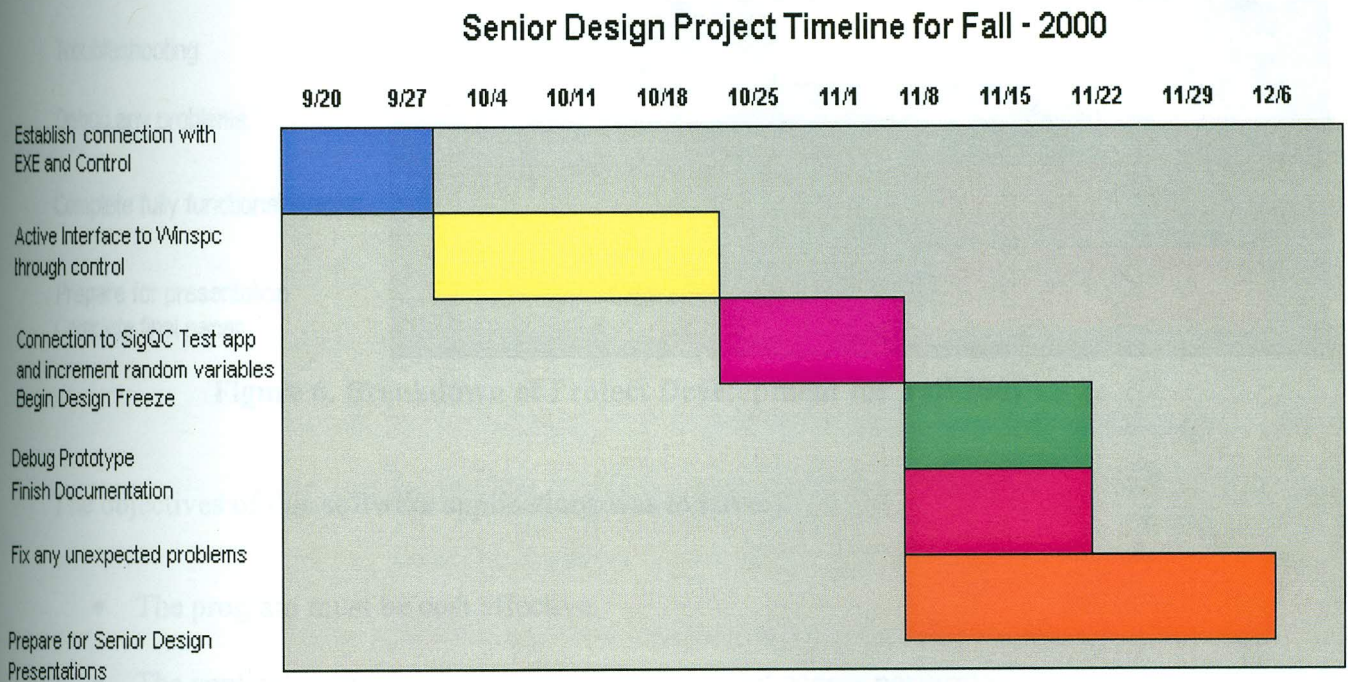


Figure 5. Breakdown of Project Development for Fall 2000

5. Proof of Design

Upon completion

Connect to SigQC Application
Test With Pulse System
Adjust to any Trico
Specifications
Troubleshooting
Debug any problems
Complete fully functional system
Prepare for presentation
Complete final paper

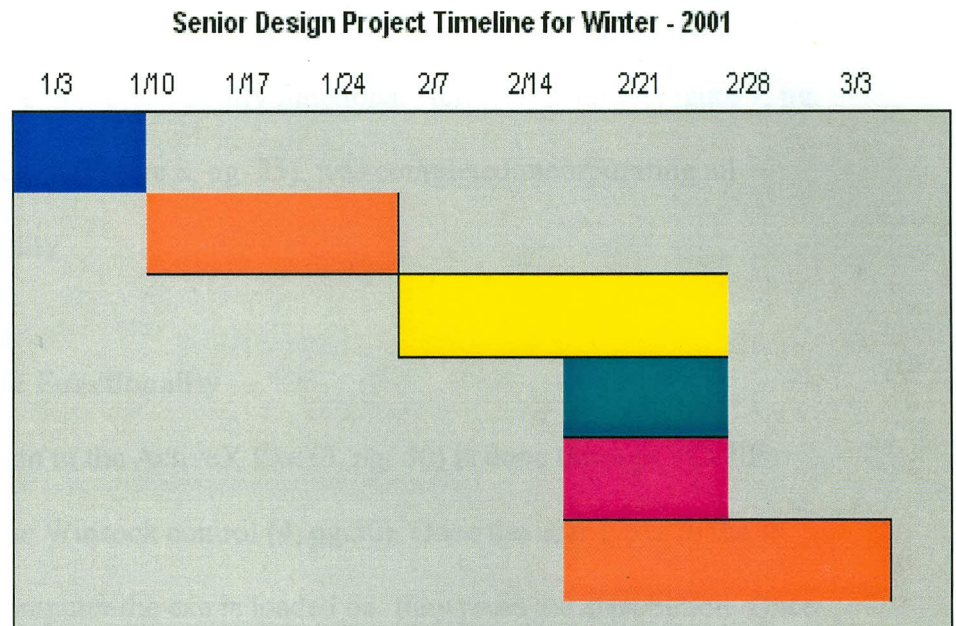


Figure 6. Breakdown of Project Development for Fall 2001

The objectives of this software applications was to have:

- The program must be cost effective
- The application needs to provide effective instant access performance without interference to SigQC application speed.
- Cross Platform Application
- Reliable Data Transfer
- The SPC software application must be compatible with SigQC

6. Proof of Design

Upon completion of this project, a fully functional ActiveX Control (Figure 7, pg. 17), along with an ActiveX exe (Figure 8, pg. 25), was completed incorporating all objectives into its functionality

ActiveX Control Complete Functionality

Connection - The connection to the ActiveX Exe (3, pg. 50) is done through TCP/IP protocol (6, pg.51), using the Winsock control (4, pg.50). Once the user types in the IP address (5, pg .51) of the computer the exe is loaded on, they press the start button. Once the message is received by the ActiveX exe, a message is sent back to the ActiveX control to confirm the connection. The Active X exe then begins telling WinSpc to open and begin creating the test case. This includes the test name, part name and collection plan. The ActiveX server sends back a string message to the Active X control, which is displayed, in a text box. The textbox on the control allows the user to see the status of WinSpc as it creates the part and test information. Once the information has been taken from SigQC, and all the testing information is created in WinSpc, a message is sent back that the test was created successfully.

Export - The export button allows for the variables being generated from SigQC to be sent. A message, along with the value, is sent to the Active X exe. Once the value is sent, another message is sent along with a variable. This second message breaks down data into accepted or rejected variables. Limits are previously set through SigQC, which defines whether data is rejected or accepted. Once variables are being received by the

Active X exe, a message is sent back to the ActiveX control (3, pg.50) telling it that the variables are being received. A message is then displayed in the textbox that the variables are being received successfully.

Halt - When the production line is halted a message is sent to the Active X exe stating that the production line is stopped. When the message is received, it sends a message back to the control, a message is then displayed in the text box that the production line is halted. At this time the Export function can be restarted and the variables are sent again.

Shutdown - If the production line is shutdown, a message box appears letting the user know this will break the connection. If yes is selected, a message is sent to the ActiveX exe. Once this message is received, a message is sent back alerting the user that the connection has been broken. The TCP/IP connection is then broken. By pressing start a connection is made again and a new test case is created.

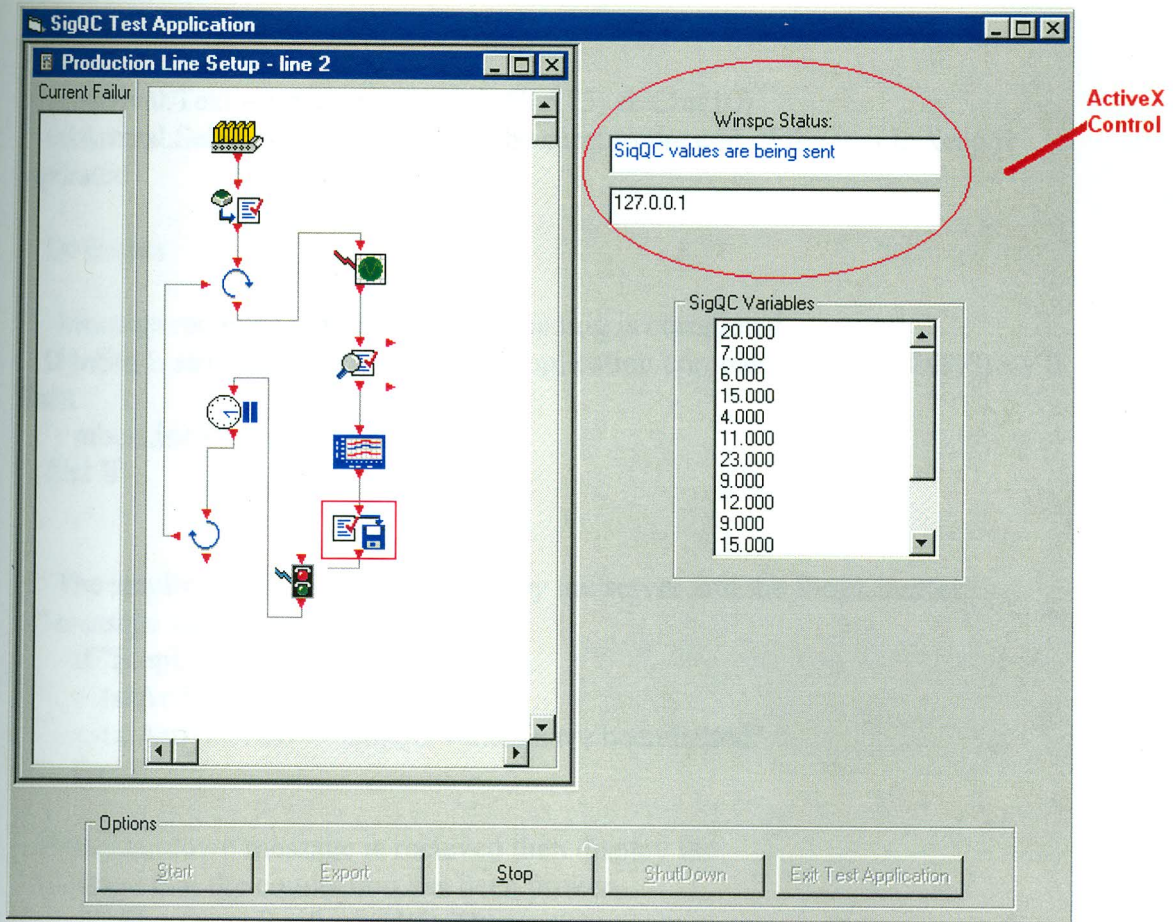


Figure 7. SigQC Test Application and ActiveX Control

The following code was used in designing the ActiveX control:

Option Explicit

```
Private mbolLineStarted As Boolean
Dim mIncrementPass As Long
Dim mIncrementFail As Long
Dim iStringLen As String
Dim iRemain As Integer
Dim strFinal As String
```

```
Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
'Display status sent from Server in textbox
Dim strdata As String
```



```

tcpClient.GetData strdata
txtArrival.Text = strdata
txtArrival.Text = txtArrival.Text & Chr(13) & Chr(10)
txtArrival.SelStart = 32000 'move the caret to the end of the text to keep it
seperated

```

```

DoEvents

```

```

' Message recieved when Conn and loading is completed
If InStr(1, strdata, "Active X Server application completed successfully") <> 0
Then
    mbolLineStarted = True
End If

```

```

' The stopline message was recieved by the server and the StopLineRec
' message was sent back to the client5
If "StopLineRec" = strdata Then
    txtArrival.Text = ""
    txtArrival.Text = "SiqQC values have been halted"
End If

```

```

'If the shutdown message is recieved then display the
'message that the connection has been broken
If "ShutDownRec" = strdata Then
    txtArrival.Text = ""
    txtArrival.Text = "The connection has been broken"
    tcpClient.Close
End If

```

```

End Sub

```

```

Public Function ExportLineStarting() As Boolean 'i_lineName As String

```

```

' When the message is recieved that the
' line is starting make connection with Server
tcpClient.RemoteHost = Text1.Text 'Change to set to whatever port it is located on
tcpClient.RemotePort = 1001 ' Remote port number

```

```

'Invoke the Connect method
tcpClient.Connect

```

```

DoEvents

```

```

' After the connection has been made send message

```

'to Open

```
Dim strdata As String  
strdata = "RunMenu"
```

```
'BEGIN ADD SPACE CODE  
iStringLen = 0  
iStringLen = Len(strdata)  
iRemain = 20 - iStringLen  
strFinal = strdata & Space(iRemain)  
'END ADD SPACE CODE
```

```
tcpClient.SendData (strFinal)  
'tcpClient.SendData (strdata)
```

```
mbolLineStarted = False  
Do Until mbolLineStarted = True  
    DoEvents  
Loop
```

End Function

Public Function ExportLineStopping() As Boolean

' If the message has been sent that the production
' line is halted. Stop variables from being sent.

```
ExportLineStopping = True
```

```
Dim strdata As String
```

```
'send message values have been halted  
strdata = "StopLine"
```

```
'BEGIN ADD SPACE CODE  
iStringLen = 0  
iStringLen = Len(strdata)  
iRemain = 20 - iStringLen  
strFinal = strdata & Space(iRemain)  
'END ADD SPACE CODE
```

```
tcpClient.SendData (strFinal)
```

```
'tcpClient.SendData (strdata)
```

End Function

```
Public Function ExportInspectData(i_product As String, i_test As String, i_cases As Variant, i_states As Variant, i_data As Variant, i_domUnits As String, i_rangeUnits As String) As Boolean
```

```
' Breakdown random data into variables to be sent to  
' Send Random data from sigQc exe to active X control
```

```
Dim strValue As String  
Dim strBuffer As String
```

```
strValue = Format(i_data, "0.000")  
strBuffer = "NewValue," & strValue
```

```
'BEGIN ADD SPACE CODE  
iStringLen = 0  
iStringLen = Len(strBuffer)  
iRemain = 20 - iStringLen  
strFinal = strBuffer & Space(iRemain)  
'END ADD SPACE CODE
```

```
tcpClient.SendData (strFinal)
```

```
'tcpClient.SendData (strBuffer)
```

```
'Display message when values are being sent  
ExportInspectData = True  
txtArrival.Text = ""  
txtArrival.Text = "SiiQC values are being sent"  
End Function
```

```
'This is a function added in order for to know  
'the difference between the line being shutdown  
'and values not being sent
```

```
Public Function ShutDown() As Boolean
```

```
ShutDown = True  
Dim strdata As String
```

```
'send message production line has been shutdown
```



```

strdata = "ShutDown"

'BEGIN ADD SPACE CODE
iStringLen = 0
iStringLen = Len(strdata)
iRemain = 20 - iStringLen
strFinal = strdata & Space(iRemain)
'END ADD SPACE CODE

tcpClient.SendData (strFinal)

'tcpClient.SendData (strdata)
DoEvents
End Function

'New Code for ICQ Watch 1-01

Private Sub Timer1_Timer()
'Display Connection State
'Label7.Caption = tcpClient.State
End Sub

Private Sub cmdStart_Click()

' Make connection with Server
tcpClient.RemoteHost = Text1.Text 'Change to set to whatever port it is located on
tcpClient.RemotePort = 1001 ' Remote port number

'Invoke the Connect method
tcpClient.Connect

End Sub

Public Function CountFailed(i_rejected As Variant)

Dim strValue As String
Dim strBuffer As String

' Breakdown random data into variables to be sent to IRC Control
' Send data from IRC exe to IRC active X control

```

```
strValue = Format(i_rejected, "")  
strBuffer = "CountFailed," & strValue
```

```
'BEGIN ADD SPACE CODE  
iStringLen = 0  
iStringLen = Len(strBuffer)  
iRemain = 20 - iStringLen  
strFinal = strBuffer & Space(iRemain)  
'END ADD SPACE CODE
```

```
tcpClient.SendData (strFinal)
```

```
'tcpClient.SendData (strBuffer)
```

End Function

Public Function CountPassed(i_accepted As Variant)

```
Dim strValue As String  
Dim strBuffer As String
```

```
' Breakdown random data into variables to be sent to IRC Control  
' Send data from IRC exe to IRC active X control
```

```
strValue = Format(i_accepted, "")  
strBuffer = "CountPassed," & strValue
```

```
'BEGIN ADD SPACE CODE  
iStringLen = 0  
iStringLen = Len(strBuffer)  
iRemain = 20 - iStringLen  
strFinal = strBuffer & Space(iRemain)  
'END ADD SPACE CODE
```

```
tcpClient.SendData (strFinal)
```

```
'tcpClient.SendData (strBuffer)
```

End Function

```
Public Sub ExportUpdateLights(i_accepted As Variant, i_rejected As Variant)
'i_labels As Variant
' Breakdown values to be sent to specified Sub
```

```
    Dim i_data As Variant
```

```
    ' Add the rejected and accepted data together to
    ' get the total amount of values sent
    Dim i_Total As Variant
```

```
    ' If the value is within accepted limits then send the
    ' accepted data to CountPassed
    If i_accepted <> 0 Then
        mIncrementPass = mIncrementPass + 1
        i_data = i_accepted
        CountPassed (mIncrementPass)
```

```
    Else
```

```
        ' If the value is within rejected limits then send the
        ' accepted data to CountFailed
        If i_rejected <> 0 Then
            mIncrementFail = mIncrementFail + 1
            i_data = i_rejected
            CountFailed (mIncrementFail)
```

```
        End If
```

```
    End If
```

```
End Sub
```

```
Public Sub ExportLineStart(i_lineName As String)
```

```
    Dim i_data As String
```

```
    ' Get the name of the line that is being tested
    'from SigQc
    i_data = i_lineName
```

```
    ' Send the name of the line that is being tested to the Server.exe
    tcpClient.SendData (i_data)
End Sub
```


6.2 ActiveX Exe Functionality

Connection - The server is always set to listen in order to know when a connection is being activated. Once the message is received that a connection is being made, the connection status label will display the connection status activity.

Open - Once the message is received to start WinSpc and create its test case, the "Run Menu" function begins. First, this function automatically loads the password. Because this is normally hand typed in WinSpc, the ActiveX server has been designed to automatically tell WinSpc the password, but only for the Data Collection where the variables are displayed. This allows for administrative rights to be set in order to access other areas of WinSpc. This event steps through creating test cases, setting limits, part names, test names and the Data Collection Name. While this is taking place, a text box displays the status of WinSpc.

Error handling - If for any reason the "Run Menu" event is unable to execute correctly, a message will be sent to The ActiveX control allowing the user to see that the event has failed.

Export Data - Once the "Enter Value" message is received, the ActiveX Server steps through the "Enter Value" Event. This event takes the variable sent and inputs it into WinSpc data collections where it is then displayed in graphical format. Once the variable is sent to the Data Collection, it is then separated into an accepted or rejected function. The rejected and accepted functions allow for the numbers to be counted and incremented

into text boxes. This allows for the user to see how many parts have failed and passed. Textboxes located on the ActiveX exe allow for this information to be displayed. A total count is also displayed in a textbox. All information is updated as it is received from the ActiveX Control. A failure count list box is also displayed on the ActiveX exe. As data is being received, the list box displays whether it has passed or failed, the test name, the date and the time the part was tested. This allows for the production line monitor to better monitor the accepted and rejected units (Figure 8.pg.26)

Halt - When the message is received that the Production Line has been halted, a message is sent back that the message was received. The status textbox then displays that the production line has been halted. Because all variables received are stored in a buffer, all numbers are entered without losing any information.

Shutdown - When the message is received that the Production Line has been Shutdown, a message is sent back that the message was received. The status textbox then displays that the connection has been lost. Because the connection status is set to listen, a connection can easily be re-established by clicking the start button located on the ActiveX control. This allows for a new test to be created

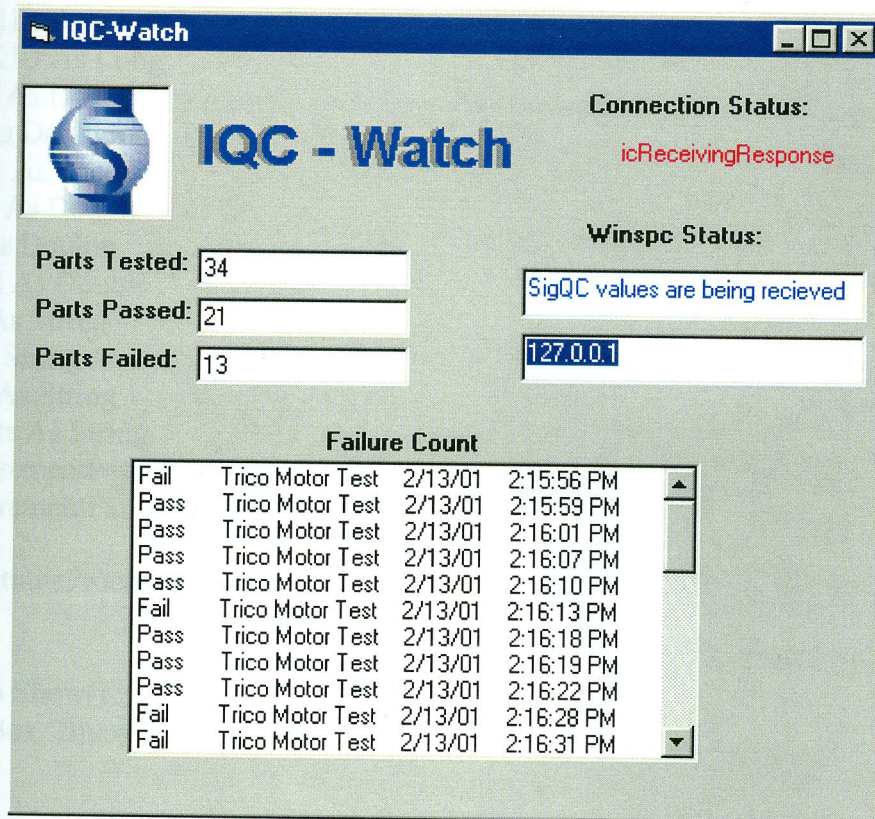


Figure 8. ActiveX Exe receiving information form ActiveX Control

The following code was used in designing the ActiveX control:

Option Explicit

'Declare all variables to be used

Dim PartObj As Object

Dim CPObj As Object

Dim DCObj As Object

Dim DBObj As Object

Dim CWobj As Object

Dim sUserName As String

Dim sPassword As String

Dim sPart As String

Dim sVar1 As String

Dim sVar2 As String

Dim sCollPlan As String

Dim Var1ID As Long

Dim PartID As Long

Dim Var2ID As Long

Dim CollPlanID As Long


```

Dim Step1ID As Long
Dim Step2ID As Long
Dim Mu1 As Double
Dim S1 As Double
Dim Cpk1 As Double
Dim Mu2 As Double
Dim S2 As Double
Dim Cpk2 As Double
Dim Mu As String
Dim S As String
Dim Cpk As String
Dim strdata As String
Dim mIncrementPass As Long
Dim mIncrementFail As Long

Dim bContinueProcess As Boolean

```

```

Public Sub Show()
    MsgBox "Show"
End Sub

```

```

Public Sub Runmenu()
' Function to begin opening . Called for sending
' data to active X exe.

```

```

sUserName = "admin"
sPassword = ""
sPart = "Trico Motor"
sVar1 = "Uniform(0, 1)"
sVar2 = "Uniform(1.7, 0.4)"
sCollPlan = "SigQC Test App"

```

```

'Create each OLE object; this will start if it's not running.
' must be found in the path or the same directory as the OLE client
'application. Otherwise, the OLE client application must start
'manually by calling Shell().

```

```

ShowStatus ("Creating Active X automation objects")

```

```

If Connect = False Then
    ShowStatus ("Failed to create Active X automation objects")
Exit Sub

```

End If

'check if is ready for login; this only needs to be done once after connecting
ShowStatus ("Checking if is ready for login")

While PartObj.ReadyForLogin <> "T"
Wend

'login to part interface
ShowStatus ("Login to part interface")

If LoginPart(sUserName, sPassword) = False Then
 ShowStatus ("Login to part interface Failed")
 LogoutPart
 Exit Sub
End If

'create a part in the root part folder
ShowStatus ("Creating a part named " & sPart)
PartID = CreatePart(sPart, 0) '0 = root part folder
If PartID < 1 Then 'part ID must be greater than 0

 ShowStatus ("Failed to create part")
 LogoutPart
 Exit Sub
End If

'create a variable characteristic for the part just created
ShowStatus ("Creating 1st variable characteristic named " & sVar1)
Var1ID = CreateVariable(sVar1, PartID)
If Var1ID < 1 Then 'variable ID must be greater than 0
 ShowStatus ("Failed to create 1st variable characteristic")
 LogoutPart
 Exit Sub
End If

'set properties for 1st variable characteristic

ShowStatus ("Setting properties for 1st variable characteristic")
If SetVar1Properties = False Then
 ShowStatus ("Failed to set properties for 1st variable characteristic")
 LogoutPart
 Exit Sub
End If

```

'create another variable characteristic for the part
ShowStatus ("Creating 2nd variable characteristic named " & sVar2)
Var2ID = CreateVariable(sVar2, PartID)
If Var2ID < 1 Then 'variable ID must be greater than 0
    ShowStatus ("Failed to create 2nd variable characteristic")

    LogoutPart
    Exit Sub
End If

'set properties for 2nd variable characteristic
ShowStatus ("Setting properties for 2nd variable characteristic")
If SetVar2Properties = False Then
    ShowStatus ("Failed to set properties for 2nd variable characteristic")
    LogoutPart
    Exit Sub
End If

'login to collection plan interface
ShowStatus ("Login to collection plan interface")
If LoginCP(sUserName, sPassword) = False Then
    ShowStatus ("Failed to login to collection plan interface")
    LogoutCP
    Exit Sub
End If

'Create a collection plan in the root collection plan folder

ShowStatus ("Creating a collection plan named " & sCollPlan)
CollPlanID = CreateCollectionPlan(sCollPlan, 0)
If CollPlanID < 1 Then 'collection plan ID must be greater than 0
    ShowStatus ("Creation of collection plan Failed!")
    LogoutCP
    Exit Sub
End If

'Create a step for the collection plan just created
ShowStatus ("Creating 1st collection plan step")
Step1ID = CreateCollectionPlanStep(CollPlanID)
If Step1ID < 1 Then 'step ID must be greater than 0

    ShowStatus ("1st collection plan step creation Failed")
    LogoutCP
    Exit Sub
End If

```



```

'Set properties for 1st collection plan step
ShowStatus ("1st collection plan step properties being set")
If SetStep1Properties = False Then
    ShowStatus ("1st collection plan step properties setup Failed!")
    LogoutCP
    Exit Sub
End If

'Create another step for the collection plan
ShowStatus ("Collection plan step being created")

Step2ID = CreateCollectionPlanStep(CollPlanID)
If Step2ID < 1 Then 'step ID must be greater than 0
    ShowStatus ("Failed to create 2nd collection plan step")
    LogoutCP
    Exit Sub
End If

'Set properties for 2nd collection plan step
ShowStatus ("Setting properties for 2nd collection plan step")
If SetStep2Properties = False Then
    ShowStatus ("Failed to set properties for 2nd collection plan step")
    LogoutCP

    Exit Sub
End If

'Login to data collection interface
ShowStatus (" data collection interface Login")

If LoginDC(sUserName, sPassword) = False Then
    ShowStatus ("Data collection interface login Failed!")
    LogoutDC
    Exit Sub
End If

'Load collection plan
ShowStatus ("Collection plan loading")
If LoadCollectionPlan(CollPlanID) = False Then
    ShowStatus ("Collection plan loading failed!")
    LogoutDC
    Exit Sub
End If

'collect data

```

ShowStatus ("Data being collected")

If CollectData = False Then

 ShowStatus ("Failed to collect data")

 LogoutDC

 Exit Sub

End If

'Login to collection plan interface

ShowStatus ("Login to collection plan interface")

If LoginCP(sUserName, sPassword) = False Then

 ShowStatus ("Failed to login to collection plan interface")

 LogoutCP

 Exit Sub

End If

'Login to data browse interface

ShowStatus ("Login to data browse interface")

If LoginDB(sUserName, sPassword) = False Then

 ShowStatus ("Failed to login to data browse interface")

 LogoutDB

 Exit Sub

End If

'Get stats for 1st variable characteristic (Uniform 1)

ShowStatus ("Getting statistics for 1st variable characteristic")

If GetStats1 = False Then

 ShowStatus ("Failed to get statistics for 1st variable characteristic")

 LogoutDB

 Exit Sub

End If

'display stats for 1st variable characteristic (Uniform 1)

Mu = CStr(Mu1)

S = CStr(S1)

Cpk = CStr(Cpk1)

ShowStatus (sVar1 & ": Mu=" & Mu & ", S=" & S & ", Cpk=" & Cpk)

'Get stats for 2nd variable characteristic (Uniform 2)

ShowStatus ("Getting statistics for 2nd variable characteristic")

If GetStats2 = False Then

 ShowStatus ("Failed to get statistics for 2nd variable characteristic")

 LogoutDB

```

Exit Sub
End If

'Display stats for 2nd variable characteristic (Uniform 2)
Mu = CStr(Mu2)
S = CStr(S2)
Cpk = CStr(Cpk2)
ShowStatus (sVar2 & ": Mu=" & Mu & ", S=" & S & ", Cpk=" & Cpk)

'logout from data browse interface
ShowStatus ("Logout from Data Browse Interface")
If LogoutDB = False Then
    ShowStatus ("Failed to logout from Data Browse Interface")
Exit Sub
End If

'Everything completed successfully

ShowStatus ("Active X Server application completed successfully")
End Sub

Private Sub ShowStatus(txt As String)

    'Send data to Client textbox to display status of
    tcpServer.SendData (txt)
    DoEvents ' Allows for data to be sent each time ShowStatus is called

    'Display status of on text in Server
    txtStatus.Text = txtStatus.Text & txt & Chr(13) & Chr(10)
    txtStatus.SelStart = 32000 'move the caret to the end of the text
End Sub

Private Function Connect() As Boolean
    Connect = False

    'Create OLE automation object for part interface
    Set PartObj = CreateObject("32.PartAuto")

    'Create OLE automation object for collection plan interface
    Set CPObj = CreateObject("32.CollectionPlanAuto")

    'Create OLE automation object for data collection interface
    Set DCObj = CreateObject("32.DataCollectionAuto")

```



```
'Create OLE automation object for data browse interface
Set DBObj = CreateObject("32.DataBrowseAuto")
```

```
Connect = True
End Function
```

```
Private Function LoginPart(Name As String, PW As String) As Boolean
```

```
'Login to part interface
LoginPart = False
```

```
PartObj.UserName = Name
PartObj.Password = PW
```

```
'check UserID to see if login was successful
If PartObj.UserID <> 0 Then
    LoginPart = True
End If
```

```
End Function
```

```
Private Function LoginCP(Name As String, PW As String) As Boolean
```

```
'login to collection plan interface
LoginCP = False
```

```
CPObj.UserName = Name
CPObj.Password = PW
```

```
'Check UserID to see if login was successful
If CPObj.UserID <> 0 Then
    LoginCP = True
End If
```

```
End Function
```

```
Private Function LoginDC(Name As String, PW As String) As Boolean
```

```
'login to data collection interface
LoginDC = False
```

```
DCObj.UserName = Name
DCObj.Password = PW
```

```
'Check UserID to see if login was successful
If DCObj.UserID <> 0 Then
    LoginDC = True
End If
```

```
End Function
```

```
Private Function LoginDB(Name As String, PW As String) As Boolean
```

'Login to data browse interface

LoginDB = False

DBObj.UserName = Name

DBObj.Password = PW

'Check UserID to see if login was successful

If DBObj.UserID <> 0 Then

 LoginDB = True

End If

End Function

Private Function LogoutPart() As Boolean

 'Logout from part interface

 LogoutPart = False

 PartObj.UserName = ""

 LogoutPart = True

End Function

Private Function LogoutCP() As Boolean

 'Logout from collection plan interface

 LogoutCP = False

 CPObj.UserName = ""

 LogoutCP = True

End Function

Private Function LogoutDC() As Boolean

 'Logout from data collection interface

 LogoutDC = False

 DCObj.UserName = ""

 LogoutDC = True

End Function

Private Function LogoutDB() As Boolean

 'Logout from data browse interface

 LogoutDB = False

 DBObj.UserName = ""

 LogoutDB = True

End Function

Private Function CreatePart(Name As String, PartFolderID As Long) As Long

 CreatePart = -1

 'create part

PartObj.CreatePart Name, PartFolderID

'Check if created the part

If Left(PartObj.CommandSuccessful, 1) <> "T" Then

Exit Function

End If

'Get ID of part just created

CreatePart = PartObj.CurrentPartID

If CreatePart < 1 Then

Exit Function

End If

End Function

Private Function CreateVariable(VarName As String, PartID As Long) As Long

CreateVariable = -1

'Create variable characteristic

PartObj.CreateVariable VarName, PartID

'Check if created the variable characteristic

If Left(PartObj.CommandSuccessful, 1) <> "T" Then

Exit Function

End If

'Get ID of variable characteristic just created

CreateVariable = PartObj.CurrentCharID

If CreateVariable < 1 Then

Exit Function

End If

End Function

Private Function SetVar1Properties() As Boolean

' Set first testing section (top)

SetVar1Properties = False

PartObj.SubgroupSize = 2 ' amount of data collections tests in one testing procedure ex
Candy test has 10 sub tests

PartObj.USL = 2 'Upper Limits

PartObj.LSL = -2 'Lower limits

PartObj.DisplayPrecision = 8 'Number of decimal digits

SetVar1Properties = True

End Function

Private Function SetVar2Properties() As Boolean

' Set second testing section(bottom)

SetVar2Properties = False

PartObj.SubgroupSize = 2

PartObj.USL = 2.5

PartObj.LSL = 0.9

PartObj.DisplayPrecision = 8

SetVar2Properties = True

End Function

Private Function CreateCollectionPlan(Name As String, CollPlanFolderID As Long) As Long

CreateCollectionPlan = -1

'create collection plan

CPObj.CreateCollectionPlan Name, CollPlanFolderID

'check if created the collection plan

If Left(CPObj.CommandSuccessful, 1) <> "T" Then

Exit Function

End If

'Get ID of collection plan just created

CreateCollectionPlan = CPObj.CurrentCollectionPlanID

If CreateCollectionPlan < 1 Then

Exit Function

End If

End Function

Private Function CreateCollectionPlanStep(CollPlanID As Long) As Long

CreateCollectionPlanStep = -1

'Create collection plan step

CPObj.CreateCollectionPlanStep (CollPlanID)

'Check if created the collection plan step

If Left(CPObj.CommandSuccessful, 1) <> "T" Then

Exit Function

End If

'Get ID of collection plan step just created

CreateCollectionPlanStep = CPObj.CurrentCollectionPlanStepID

```
If CreateCollectionPlanStep < 1 Then
    Exit Function
End If
End Function
```

```
Private Function SetStep1Properties() As Boolean
```

```
'Uniform Test 1
```

```
SetStep1Properties = False
```

```
CPObj.StepNumber = 1 'StepNumber property is 1-based
```

```
CPObj.CharType = "V" 'This property returns the current characteristic's type Possible
values are "V" or "A".
```

```
CPObj.CharID = Var1ID 'This property returns the ID value for the current
characteristic.
```

```
CPObj.SampleSize = 2 'This property returns the sample size for the current
characteristic.
```

```
SetStep1Properties = True
```

```
End Function
```

```
Private Function SetStep2Properties() As Boolean
```

```
'Uniform Test 2
```

```
SetStep2Properties = False
```

```
CPObj.StepNumber = 2 'StepNumber property is 1-based
```

```
CPObj.CharType = "V"
```

```
CPObj.CharID = Var2ID
```

```
CPObj.SampleSize = 2
```

```
SetStep2Properties = True
```

```
End Function
```

```
Private Function LoadCollectionPlan(CollPlanID As Long) As Boolean
```

```
'Load collection plan
```

```
LoadCollectionPlan = False
```

```
DCObj.ProcedureID = CollPlanID
```

```
While DCObj.ProcedureID <> CollPlanID
```

```
Wend
```

```
LoadCollectionPlan = True
```

```
End Function
```

```
Private Function CollectData() As Boolean
Dim i As Integer
```

```
CollectData = False
```

```
If lstValue.ListCount > 0 Then
    For i = 0 To lstValue.ListCount - 1
        'CurrentStep property is 0-based
        DCObj.CurrentStep = 0
        DCObj.Value = lstValue.List(i)
```

```
    Next i
```

```
Else
```

```
    For i = 0 To lstValue.ListCount - 1
        'CurrentStep property is 0-based
        DCObj.CurrentStep = 0
        DCObj.Value = 0
```

```
    Next i
```

```
End If
```

```
'Second variable set
```

```
'CurrentStep property is 0-based
```

```
If lstValue.ListCount > 0 Then
```

```
    For i = 0 To lstValue.ListCount - 1
        'CurrentStep property is 0-based
        DCObj.CurrentStep = 1
        DCObj.Value = lstValue.List(i)
```

```
    Next i
```

```
Else
```

```
    For i = 0 To lstValue.ListCount - 1
        'CurrentStep property is 0-based
        DCObj.CurrentStep = 1
        DCObj.Value = 0
```

```
    Next i
```

```
CollectData = True
```

```
End If
```

```
End Function
```

```
Private Function UnloadCollectionPlan() As Boolean
```

```
'Close out collection plan
```



```
Dim CurrentCollPlanID As Long
UnloadCollectionPlan = False
```

```
'unload collection plan
DCObj.ProcedureID = 0
While DCObj.ProcedureID <> 0
Wend
```

```
UnloadCollectionPlan = True
End Function
```

```
Private Function GetStats1() As Boolean
' Test information for Uniform Test one
GetStats1 = False
```

```
DBObj.CurrentCharType = "V"
DBObj.CurrentCharID = Var1ID
DBObj.CurrentSubGroupNumber = 100
Mu1 = DBObj.Average
S1 = DBObj.StandardDeviation
Cpk1 = DBObj.Cpk
```

```
GetStats1 = True
```

```
End Function
```

```
Private Function GetStats2() As Boolean
'Information for Uniform test 2
GetStats2 = False
```

```
DBObj.CurrentCharType = "V"
DBObj.CurrentCharID = Var2ID
DBObj.CurrentSubGroupNumber = 100
Mu2 = DBObj.Average
S2 = DBObj.StandardDeviation
Cpk2 = DBObj.Cpk
```

```
GetStats2 = True
```

```
End Function
```

'TCP/IP code

Private Sub tcpServer_Close()

'If connection has been broken and is re-opened, set to listen

'If SigQC has been shutdown allow for reconnection to be made when reopened

tcpServer.Close

UserControl_Initialize

End Sub

Private Sub tmrDataGetter_Timer()

'This subroutine will Poll the TCP Socket

'to check for data left in the buffer.

'It is activated by the TCP_OnDataArrival

'It is deactivated when there is no data left in the pool

'or the stop command.

'

'Also, it will not process any data if a stop

'command has been recieved

'and will process data again if the start command

'is recieved.

'Declare a variable for the incoming data.

'Invoke the GetData method to run data from control

Dim strdata As String

Dim strBuffer As String

Dim strCommand As String

Dim iMidPoint As Integer

Dim i_data As Variant

tcpServer.GetData strMultiCmd, vbString, bytesTotal

tcpServer.GetData strBuffer, vbString, 20

If strBuffer = "" Or strBuffer = Null Then

tmrDataGetter.Enabled = False

End If

```

'Breakdown data to commands and strData
' The buffer is broken down by the space in the middle
strBuffer = Trim(strBuffer)

iMidPoint = InStr(strBuffer, ",")

'If there is a comma in it then we need to extract
'the data that is associated with that command...
'For example, NewValue,231.000 We need to find out
'what the command is and what the data that goes along
'with the command is...

' strdata is the charactures after the break
' on the Right
If iMidPoint > 0 Then
    strdata = Mid(strBuffer, iMidPoint + 1)

    i_data = Mid(strBuffer, iMidPoint + 1) ' For IQC

' strCommand are the charactures on the
' left after the break
    strCommand = Left(strBuffer, iMidPoint - 1)
Else
    strCommand = strBuffer
End If

'Case statements to run function when
'specified command is sent

'If this is true then process the command
'If not it will skip over

Select Case strCommand
    Case "RunMenu" 'Opens
        Runmenu

    Case "DataValues" 'Enter values into list box
        DataValues

    Case "NewValue" ' Enter the new value in the textbox
        EnterValue strdata

```



```
Case "StopLine" 'message has been sent that values are no longer being sent
    StopLine
    'A stop has been encountered...
    'stop all processing any further commands until
    'a start is encountered (see before this select statement)
```

```
Case "ShutDown" 'Message has been sent that connection has been lost
    ShutDown
    tmrDataGetter.Enabled = False
```

```
Case "CountFailed"
    CountFailed (i_data)
```

```
Case "CountPassed"
    CountPassed (i_data)
```

```
End Select
```

```
DoEvents 'Now process anything else that needs to be done.
```

```
End Sub
```

```
Private Sub UserControl_Initialize()
    "Text1.Text = tcpServer.LocalIP 'For demonstration puposes only!
    ' Set the LocalPort property to an integer.
    ' Then invoke the Listen method.
    tcpServer.LocalPort = 1001
    tcpServer.Listen
    Text1.Text = tcpServer.LocalIP
End Sub
```

```
Private Sub tcpServer_ConnectionRequest(ByVal requestID As Long)
    ' Check if the control's State is closed. If not,
    ' close the connection before accepting the new
    ' connection.
    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)
    tmrDataGetter.Enabled = True
End Sub

```

```

Public Sub DataValues()
' Enter the value into and update graphical format
    CollectData
    lstValue.Clear
End Sub

```

```

Public Sub EnterValue(strNewValue As String)

    txtStatus.Text = "SigQC values are being recieved"

    ' Send recieved data to txtValue, update lstValue
    txtValue.Text = strNewValue
    lstValue.AddItem Format(Val(txtValue.Text), "0.000")
    txtValue.Text = ""

    ' Run DataValues command to enter values into Data Collection
    DataValues

End Sub

```

```

Public Sub ShutDown()
' If shut down message is sent display that the
' connection has been lost.

    txtStatus.Text = ""
    txtStatus.Text = "The connection has been broken "

```

```

DoEvents

```

```

' If shutdown message recieved, send message back to client
' that the message was recieved and to break connection
If txtStatus.Text = "The connection has been broken " Then
    strdata = "ShutDownRec"

```

```
tcpServer.SendData (strdata)
End If
```

```
End Sub
```

```
Public Sub StopLine()
Dim strdata As String
Dim StopLineRec As String
```

```
'If values are no longer being sent
'display message that Export has been halted
txtStatus.Text = ""
txtStatus.Text = "The export function has been halted"
```

```
DoEvents
```

```
'If message recieved, send message back to let client know message rec
If txtStatus.Text = "The export function has been halted" Then
    strdata = "StopLineRec"
    tcpServer.SendData (strdata)
```

```
End If
End Sub
```

```
'ICQ Watch Pass/Fail Code
```

```
Public Function CountTested(i_data As Variant)
```

```
    i_data = mIncrementFail + mIncrementPass 'Send value to CountTested
```

```
DoEvents
```

```
' Send recieved data to txtCountTested
txtTested.Text = i_data
```

```
End Function
```

```
Public Function CountFailed(i_data As Variant)
```

```
    mIncrementFail = mIncrementFail + 1
```

```
DoEvents
```

```
CountTested (mIncrementFail) 'Send value to CountTested
```



```
' Send recieved data to txtCountFailed
txtFailed.Text = i_data
List1.AddItem "CountFailed" & i_data
```

DoEvents

‘ the Chr32 allows for the columns to be separate and orderly when displayed in the listbox
this allows for the information to be in organized column format.

```
lstCount.AddItem "Fail" & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) &
Chr(32) & Chr(32) & Chr(32) & "Trico Motor Test" & Chr(32) & Chr(32) & Chr(32) &
Chr(32) & Date & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Time
```

End Function

```
Public Function CountPassed(i_data As Variant)
mIncrementPass = mIncrementPass + 1
```

DoEvents

```
CountTested (mIncrementPass) ' Send value to CountTested
```

```
' Send recieved data to txtCountPassed
txtPassed.Text = i_data
List1.AddItem "CountPassed" & i_data
```

DoEvents

‘Chr32 allows for the columns to be separate and orderly when displayed in the listbox

```
lstCount.AddItem "Pass" & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) &
Chr(32) & "Trico Motor Test" & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Date &
Chr(32) & Chr(32) & Chr(32) & Chr(32) & Time
```

End Function

```
Private Sub tmrStatus_Timer()
```

```
'Display text form of Wisock connection state
```

```
If tcpServer.State = 4 Then
```

```
    lblStatus.Caption = "Connected"
```

```
Else
```

```
    lblStatus.Caption = "Not Connected"
```

```
end if
```

7. Conclusion

I selected the data transfer project because it covered the subjects of Networking and Programming. I wanted to enhance my experience as a software developer. I also wanted to make use of programming and networking skills that I have acquired over the last few years. This data transfer project was an ideal project to make use of the various skills that I acquired while pursuing my bachelor's degree in Information Engineering Technology and the industry experience I had acquired during my co-op.

The following were the hardware requirements for the project:

- Client Machine
- Server Machine

The Following is the list of various software tools and technologies I used to complete the project:

- ActiveX controls
- ActiveX exe
- SPC packets
- WinSpc
- Visual Basic Programming Software

The following is a table below that shows the various skills I acquired in different areas by taking different courses while pursuing my Bachelor's degree in Information Engineering Technology.

Area	Technology	<u>Courses</u>
Networking	TCP/IP Connection LAN Networking	Networking I, II, III
Programming	Visual Basic Programming	Object Oriented Programming I, II, III
Graphical Design	Graphic Design	Multimedia Production Graphics for Presentations and Publication
Documentation	Proposal/Final reports	Technical Writing I, and II

In short, I utilized various skills I acquired by taking a number of courses to develop my senior design project.

I faced many challenges during the development of this program. First of all I have never developed a project that was this large especially by myself. My experience in programming was very basic. At the beginning of this project I had only taken Object Oriented programming I and II. I had no experience with ActiveX Controls and ActiveX exe. Once I completed Object Oriented III, two quarters of my co-op and read various books on Visual basic programming I had a better knowledge, which helped me understand the project and complete my goals.

I only faced two major problems throughout my project. The first was through Trico. Customer specifications were constantly being changed during the design of the working prototype. This caused problems with not only the time factor but also with having to change my timeline along with my explanation of the project. After Senior Design II all specifications were kept the same and the customer was happy with the product prototype.

The second problem that I faced was coming to understand WinSpc along with its ability to receive event commands. Datanet (13, pg.52) does not offer much information when it comes to making an OLE connection to WinSpc. A small amount of information in the help file along with looking at the object browser was all the information I had to begin designing the product. This turned out to be the most time consuming and stressful part of this project. Once I had a grasp on how the connections can be made the project began to move along at a steady pace. During the rest of the project I only faced minor troubleshooting problems.

Notes

1. **Client:** part of client-server architecture. Typically, a client is an application that runs on a personal computer or workstation and relies on a server to perform some operations
2. **Server:** A computer or device on a network that manages network resources. For example, a file server is a computer and storage device dedicated to storing files. Any user on the network can store files on the server. A print server is a computer that manages one or more printers, and a network server is a computer that manages network traffic. A database server is a computer system that processes database queries. Servers are often dedicated, meaning that they perform no other tasks besides their server tasks. On multiprocessing operating systems, however, a single computer can execute several programs at once. A server in this case could refer to the program that is managing resources rather than the entire computer.
3. **ActiveX Control/ Active X Exe:** ActiveX is not a programming language, but rather a set of rules for how applications should share information. Programmers can develop ActiveX controls in a variety of languages, including C, C++, Visual Basic, and Java. ActiveX controls have full access to the Windows operating system. This gives them much more power than Java applets, but with this power comes a certain risk that the applet may damage software or data on your machine. To control this risk, Microsoft developed a registration system so that browsers can identify and authenticate an ActiveX control before downloading it. Another difference between Java applets and ActiveX controls is that Java applets can be written to run on all platforms, whereas ActiveX controls are currently limited to Windows environments.
4. **Winsock Control:** Short for Windows Socket, Winsock is an Application Programming Interface (API) for developing Windows programs that can communicate with other machines via the TCP/IP protocol. Windows 95 and Windows NT comes with Dynamic Link Library (DLL) called *winsock.dll* that implements the API and acts as the glue between Windows programs and TCP/IP connections. In addition to the Microsoft version of *winsock.dll*, there are other freeware and shareware versions of *winsock.dll*. However, there is no official standard for the Winsock API, so each implementation differs in minor ways.
5. **IP Address:** An identifier for a computer or device on a TCP/IP network. Networks using the TCP/IP protocol route messages based on the IP address of the destination. The format of an IP address is a 32-bit numeric address written as four numbers separated by periods. Each number can be zero to 255. For example, 1.160.10.240 could be an IP address. Within an isolated network, you can assign

IP addresses at random as long as each one is unique. However, connecting a private network to the Internet requires using registered IP addresses (called Internet addresses) to avoid duplicates.

6. **TCP/IP Protocol:** Abbreviation for Transmission Control Protocol/Internet Protocol, the suite of communications protocols used to connect hosts on the Internet. TCP/IP uses several protocols, the two main ones being TCP and IP. TCP/IP is built into the UNIX operating system and is used by the Internet, making it the de facto standard for transmitting data over networks. Even network operating systems that have their own protocols, such as Netware, also support TCP/IP.
7. **Data:** Distinct pieces of information usually formatted in a special way. All software is divided into two general categories: data and programs. Programs are collections of instructions for manipulating data. Data can exist in a variety of forms -- as numbers or text on pieces of paper, as bits and bytes stored in electronic memory, or as facts stored in a person's mind.
8. **UDP Protocol:** Short for User Datagram Protocol, a connectionless protocol that, like TCP, runs on top of IP networks. Unlike TCP/IP, UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network. It's used primarily for broadcasting messages over a network.
9. **Visual Basic Programming Language:** A programming language and environment developed by Microsoft. Based on the BASIC language, Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces. Instead of worrying about syntax details, the Visual Basic programmer can add a substantial amount of code simply by dragging and dropping *controls*, such as buttons and dialog boxes, and then defining their appearance and behavior.
10. **C++ Programming Language:** A high-level programming language developed by Bjarne Stroustrup at Bell Labs. C++ adds object-oriented features to its predecessor; C. C++ is one of the most popular programming languages for graphical applications, such as those that run in Windows and Macintosh environments.
11. **Cross Platform Application:** Refers to the capability of software or hardware to run identically on different platforms. Many applications for Windows and the Macintosh, for example, now produce binary-compatible files, which means that users can switch from one platform to the other without converting their data to a

new format. Cross-platform computing is becoming increasingly important as local-area networks become better at linking machines of different types.

12. **DataNet** - a division of the Detroit Gauge Group, a family of precision tooling, gauging, testing equipment and electronic probe manufacturers established in 1941. This half-century legacy in hands-on metrology and manufacturing has greatly influenced the design of our SPC products to promote ease of use and practicality in real-world applications.
13. **WinSpc Software:** Technologically advanced statistical process control application
14. **Event:** An action or occurrence detected by a program. Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory.

References

1. Knudson, Richard. Managing Large Scale Development Basic. Microsoft Corporation. 1995
2. Maloney, Jim. Distributed COM, Application Development using VB 6. UCI Corporation. 1999
3. Alvarez, Steve, Purshottam Chandak, Ramesh Chandak, and Gene Swartzfager. Visual Basic 6 Object Oriented Programming Gold Book. Coriolis Technology Press. 1999
4. Boutquin, Pierre, David Jung, Andrew J. Indovina, and Mark Spenik. Visual Basic 6 InterActixX Course. Wait Group Press. 1998
5. Boutquin, Pierre, John Conley, Loren Eidahl, David Jung, Lowel Mauer, and Jack Purdman. Visual Basic 6 Super Bible. Sams Publishing. 1999
6. Planet Source Code. 8 May 2000. www.planet-source-code.com
7. Coleman, Neil. CEO, Signalysis, Inc. Personal Interview. May 9, 2000.
8. Ketron, Jon. Software Developer, Signalysis, Inc. Personal Interview. May 9, 2000