VB6 Software for communicating with IBC Tcp/ip readers

This document will provide a brief and simplistic introduction into using tcp/ip to communicate with IBC readers using VB6 code. Prior to reading this document and attempting to write code, it is suggested that you first read the IBC Tcp/ip Manual, and also Application Note 020, which explains how to reprogram IBC tcp/ip readers from their default ip address and port assignments.

The easy way to communicate with IBC tcp/ip readers using VB is to make use of the Winsock control. This control contains all of the necessary properties to control the connection.

To start with, create a form and add 2 Winsock controls. Name the controls tcpclient and tcpserver. For the control named tcpclient, set the remotehost property to the ip address of the reader, and set the remoteport property to the data port which is programmed in the reader. For the socket named tcpserver, set the remotehost and remoteport properties to 0. For both controls, the localport property can be set to 0. Also on your form, create four command buttons, with the captions *connect*, *listen*, *disconnect,* and *send.* Also add 2 textboxes, named text1 and text2. The controls will have the following functionality:

connect	opens a connection to a reader which is a server
listen	opens a connection to a reader which is a client
disconnect	closes either of the above connections
send	sends data to the reader
textbox	shows data returned from the reader. Call this control text1
textbox	data to be sent to the reader. Call this control text2

Reader as a server

Let's look at connecting to a reader operating as a server first. To do this, you must use the connect method of the winsock control. In this example, the winsock control is named tcpclient. We have chosen to use a boolean value (tcpconnect) and set it to true when the connection occurs. If no connection occurs after a short period, we time out. We use the windows sleep api to test the socket state every 100ms. This functionality could also be performed by using a timer. The subroutine tcpclient_connect is the Connect Event for the winsock control, which will execute when the connection has been established.

The code to establish the connection is:

```
dim tcpconnect as boolean

Private Sub tcpclient_Connect()        ' winsock connect event
tcpconnect = True                      ' tell us that the connection is made
End Sub

sub  open_tcp_server()
tcpconnect = false
on error goto bad_connect
tcpclient.connect                      ' invoke the connect method
```

```
j = 0

test_socket::
Call Sleep(100)
j = j + 1
If j > 10 Then GoTo bad_connect
If tcpconnect = False Then GoTo test_socket
text1=”connected”
exit sub

bad_connect:
text1=”no connect”
end sub
```

Set your command button *connect* to execute the sub open_tcp_server. When you click, the connect method is invoked, and the subroutine waits until a connection has been established (tcpconnect=true), or 1000ms has passed. The status is shown in the textbox *text1*.

To close the connection, invoke the close method as follows. Your button captioned *disconnect* should call this routine.

```
sub close_tcp_connections()
on error resume next
tcpclient.close
tcpconnect=false
tcpserver.close                ‘ explained later in this document
exit sub
```

If you want, you can add another command button for testing the status of the connection. Caption this command button with *status*. You can now check the status of the connection at any time by clicking on this button. The executed code should be as follows:

```
sub  check_status()
text1 = tcpclient.state
end sub
```

You can get the listing of state values by looking at the winsock documentation in VB. The most common states are:

open	1	closed	0
listening	2	connecting	6
connected	7	closing	8

Reader as a client

When the reader is set up as a client, it is the reader that initiates the connection, not the pc. To allow the pc to receive and respond to the connection request, you must invoke the listen method of the winsock control. Once the listen method is invoked, the winsock control simply waits until a connection request is received from a remote client (in this case, the reader), and lets you know when a connection request has been received by executing the connectionrequest event. You then must call the accept method to actually accept the connection. Please note that the accept method should be called on a new instance of the winsock control, so that you can accept and control multiple connection requests from the same socket, but for the purposes of this document we will use the same winsock control to accept the request.

Remember that when we tested talking to a reader which was a server, we used the winsock control on our form that was named tcpclient. We will use the other winsock control on the form, named tcpserver, to act as the socket for accepting incoming connections. Because we are accepting connections, we must also assign a port number to use for the incoming connection. This is because when the reader (acting as a client) wants to connect to us, it must specify a port number to connect to. We must prepare the pc for this connection by telling the socket what port number we expect the connection request to come in on. The default port set up in the reader is 57, so if you have not changed this, you will need to set the localport property of the socket to 57. You can do this in the code. An example follows:

```
sub setup_listen_socket()
tcpserver.localport=57              ‘ set our port to 57
tcpserver.remoteport=0
tcpserver.listen
end sub

sub tcpserver_connectionrequest(requestid as long)        ‘ winsock conectionrequest event
tcpserver.accept  requestid
text1=”accepted connection from “+tcpserver.remotehostip
end sub
```

Set up the command button which we captioned *listen* to execute the setup_listen_socket routine. The socket will listen, and when a connection request has been made and accepted, we will put the reader’s ip address in the text1 textbox. The connection is now active, and you can communicate with the reader. To close the connection, click on the *disconnect* button described earlier. This will close both the tcpclient socket and tcpserver socket.

Communicating with the reader

The methods of communicating with a reader are the same, whether the reader is a server or a client. To send data to a reader you use the senddata method of the socket. To receive data, you use the getdata method. There is also a dataarrival event, which occurs whenever new data has arrived. For our example, we will use the dataarrival event to trigger the getdata method, simply because it can run without intervention. The code to do this is shown below. We are asssuming in this example that we are set up as a client, and the reader is set up as a server (so we are using the socket named tcpclient). It is also assumed that the connection has already been established.

```
dim received_reader_data as string

sub startup_receive
received_reader_data=""
end sub

sub tcpclient_dataarrival(totalbytes as long)        ' socket data arrival event, occurs when there is data
dim incomingdata as string
tcpclient.getdata incomingdata,vbstring
received_reader_data=received_reader_data+indomingdata
text1=received_reader_data
end sub
```

As you can see from the above example, when data arrives, the dataarrival event is triggered which then "gets" the data and appends it to received_reader_data, which is then copied to text1 so that you can see the received data on the form. Once you set this up, you will not have to do anything to actually read data coming in from the reader.

To senddata to the reader, we use the senddata method, such as the following:

```
tcpclient.senddata "V"+chr$(13)
```

This sends the command V followed by a carriage return (hex 0d) which is the command sent to all IBC readers to return their identification string. Since most IBC commands end with a carriage return (unless you are in protocol mode), you can use the text2 textbox on the form for typing in the command to send to the reader, and then on your command button which is captioned *send*, you could execute the following:

```
sub  send_data_to_reader()                    ‘ called from command button
text1=””                                      ‘ clear the receive box before we send the command
tcpclient.senddata trim(text2)+chr$(13)
end sub
```

Now, you can test communications with any reader, set up as a client or a server, by simply putting the reader command into the text2 textbox, and clicking on the *send* button. The response will automatically now show up in the other text box.

The preceeding examples are simplistic and are only meant as a guideline in getting started with the software required to communicate with the readers. Once you are able to test and understand the code shown here, it should be easy to develop either a client or server application for the pc which will talk with the readers.