Communicating over Ethernet to a Dalsa BOA or SPOT vision camera

Application: A customer wanted to use a Red Lion HMI in conjunction with a vision inspection camera to use as a standalone vision checking system. The customer wanted to check the barcode of a part as it passed by, and reject the barcode if it didn’t match a barcode value inputted by the HMI.

Products Used: Red Lion G307K HMI and Dalsa Boa camera with Dalsa iNspect software.

Overview
The following document will help explain the methods by which communication can be established with any device that can communicate ASCII characters over Ethernet, to a Dalsa vision camera (Boa and SPOT vision cameras).

The Dalsa Vision Cameras come equipped with several onboard communication protocols, but the most used is a method which employs scripts that run in the background as executables on some camera event—these scripts run either before an image is taken, after an image is taken, upon startup of the camera, or on a periodic basis, for which a period can be selected. The camera runs the scripts based on these events. The scripts often do procedural things like transmit, parse and format strings, evaluate and compare inputs and outputs (of variables, bits, strings, etc) and generally, anything else. This scripting is also a very powerful logic engine that can do quite a lot for a vision solution beyond just telling a user if something ‘passes’ or ‘fails’.

NOTE: a good way to test the communications of your Dalsa device while programming it, is by downloading a version of “Hyperterminal” or another modem controller that will allow you to send and receive ASCII characters over TCP/IP to a known IP address over a known port, or will allow you to listen for a message over TCP/IP FROM a known IP address, over a known port. The example listed in this document was tested with hyperterminal.

Though your application may be different, there are usually two scripts that you’ll want to create: a periodic script, that will either listen for a message from another device, or communicate a message to indicate a change of state on the camera; and a post-image process that will handle information received from a vision inspection.

Environmental Setup of the Boa
After you create a vision solution by configuring the ‘sensor’ and configuring the ‘tools’ portion of iNspect, click on the “control” button to access the controls inside the camera itself. Based on what devices you’re planning on using, you’ll have to decide if the camera is going to be a “server” or a “client”. It doesn’t really matter—at the end of the day, the devices are talking, but one has to contact the other, first.

Note that for Hyperterminal usage in testing, it’s best to set the Dalsa BOA or SPOT to ‘server’, so the hyperterminal can ‘call’ the dalsa camera. Otherwise, you’ll have to tell the hyperterminal to
‘wait for a call’ from the client dalsa camera. Either way works just fine, it’s just more confusing for the camera to be the client if you’ve only done this once.

Click the “Add TCP/IP Stream” button. Inspect wants to know whether it’s a server or client. If we’re the server, we won’t care what IP address is contacting us. If we’re the client, we need to know the IP address of the server. We also need to specify the port. Be aware that some devices use dedicated ports, so use something safe. In our configurations, we’ve used port “6010” or “5050”.

In Hyperterminal, you’ll need to create a new connection, and on the dropdown connection, select “TCP/IP” connection. It’ll ask you for a name and description, for which you can give it anything you’d like.

Click OK and you’ll arrive at a window labelled “Connect To”. Click on the Connect Using drop-down, and select TCP/IP (Winsock). Selecting this will change the fields to an IP address and a port number. Specify the IP address you set your Dalsa camera to, and the port you specified in the TCP/IP Stream settings above. Our demo Camera uses the IP address “192.168.0.2”. As mentioned before, our demos like to use port 6010 for comms with the dalsa camera.
Additionally, click on File > Properties, and click the ‘Settings’ tab. Click “ASCII Setup” and be sure to click the boxes marked ‘Send line ends with line feeds’ and ‘Echo typed characters locally’, so it’s easier to follow along on your emulation.

If all is set up correctly, you’ll see the words ‘connected’ along with an elapsed time that you’ve been connected.

**Periodic Function Setup**

Now that the Camera and your hyperterminal session are properly set up, it’s time to create a periodic function to handle incoming requests or outgoing transmissions of changes of state. The periodic function will execute fully at each interval specified in the setup. A script has to execute in order to know what to do with an incoming message on the TCP/IP buffer.

The logic behind the periodic function will look like this:

1. **Read the incoming buffer**
2. If there is information in the incoming buffer, put it into a string
3. Parse the string—that is, read specific characters at specific positions in the string, and pack them into new strings
4. If needed, convert new strings to other variable types (integers or bits for example)
5. Execute if-then statements based on the values of these new strings, integers or bits
6. Write strings or write other variables AS strings to transmit data back to device
7. Initialize variables at the end of the script

The command to read the incoming buffer and place the data into a script is the ReadString command. It has two arguments, the incoming TCP/IP port you configured (which shows up as a variable in the variable list) and the ending character of the string. The ending character of the string is usually the character for a “line feed” or “carriage return”. It’s appropriate to note that ASCII got it’s start in the printing industry, so in ASCII telecommunications, a carriage return character would prompt the printing carriage of a typewriter or teleprinter return to the start, and a line feed character would advance the paper roll by one line of type. The ASCII code for Carriage Return (CR) is 13, and line feed (LF) is 10. In hyperterminal, hitting the ‘return’ key will send a line feed. In modern usage of ASCII, line feeds and carriage returns typically are used as a signal the end of a string of text. So to read any incoming string, our example would use:

\[ \text{ReadBuffer} = \text{ReadString(TcpP6010, 10)} \]

This puts the entirety of the incoming information on the TCP connection into a string variable we just dimensioned, called “ReadBuffer”. Every time the period recycles, the incoming stream will update the ReadBuffer (if there is an incoming string).

The next part of the script is an IF-THEN statement that evaluates whether the ReadBuffer string is empty, and if it’s not, dimensions two new string variables: CommandString and CommandCharacter. CommandString is the entirety of the ReadBuffer string variable, which CommandCharacter is only the first character of the string. We capture only a particular character or character subset from another string using the Substring Function. This function has three arguments: Parent String, Starting Position of Parent String, and Length. In our example, the incoming string will always have the command character in the front part of the string, so we always want to capture the characters from the 0th position in the string, and only grab a string length of 1. In this manner, we can use one string to transmit both a trigger command (at the first position of the string), AND any other information (starting at the second position and going on as long as you need). So the IF-THEN statement that executes is this:

\[ \text{If(ReadBuffer !=""}) } \]
\[ \text{CommandString = ReadBuffer} \]
\[ \text{CommandCharacter = Substring(CommandString,0,1)} \]
\[ \text{Endif} \]

The next step looks for two conditions—if the CommandCharacter string isn’t empty, and is NOT “T” (which would indicate a trigger command for the camera), parse the CommandString variable from the second character through the 12th character, and write it to the variable “IdealCodeVar”, which is the correct code value that the vision system’s barcode tool should be seeing. This statement allows a user
to transmit a new value for the variable “IdealCodeVal” without executing a trigger command. This IF-THEN statement looks like this:

    If(CommandCharacter !="" AND Substring(CommandCharacter,0,1)!="T")
    IdealCodeVal = Substring(CommandString,1,12)
    endif

Finally, we have to handle when we DO see a trigger command. This If-THEN statement looks like this:

    If(CommandCharacter = "T")
        trigger ()
        CommandCharacter = ""
    endif

Notice that we clear out the command character after triggering; failing to do so will trigger the camera continuously (because of the logic of the above IF-THEN statements, there’s no other way to clear the CommandCharacter variable)

Post-Image Process
The post-image process handles what to do with the value of the barcode—namely, compare it to the IdealCodeVal variable we created in the periodic function.

The Post Image process compares the two values—the barcode value, and the ideal code value (IdealCodeVal), and if they are equal, allow the part to pass; if they are not, reject the part. In our application, rejecting the part meant firing an output for a given period of time, and for a given duration.

In this particular application, the customer wanted to use the Dalsa vision camera to drive a solenoid at a reject station, to push a bad part off the conveyor.

For More Information
Contact Jake Lydey at Lydey Automation at 419-893-0283 or email him at jakelydey@lydey.com