



Application Note #4434

Creating a Custom Ethernet Communication Interface to a Galil Controller

For most applications, the standard GalilTools communication library or API (Application Programming Interface) provides the best method of communicating from either a Windows or Linux computer to a Galil controller. The GalilTools communication library provides calls such as “Command()” and “ProgramDownload()” that provide a wrapper to make it easier for a programmer to get up and running in their language of choice (C#, C++, VB, LabVIEW, etc...). The GalilTools communication library handles the lower level driver functions and communication protocols so that the programmer doesn’t have to. However, there are some situations in which it is necessary or required to communicate to a Galil controller without using the GalilTools communication library. This application note will discuss what a user should be aware of when creating their own communication interface.

Part I – Standard Communication

1) Opening a Socket

Most programming languages that have an Ethernet interface have an “OpenSocket” type of function that will handle establishing the UDP or TCP/IP connection. The arguments will usually be the IP address of the device you are connecting to as well as a port number. For a standard connection to a Galil controller, the user should connect on port number 23 (Telnet). Other port numbers are allowed such as those above port 1000 when needed for special applications. See the IK command for more info.

2) Sending a Command

Once a socket is established, the user will need to send a Galil command as a string to the controller (via the opened socket) followed by a Carriage return (0x0D).

3) Receiving a Response

The controller will respond to that command with a string. The response of the command depends on which command was sent. In general, if there is a response expected such as the “TP” Tell Position command, the response will be in the form of the expected value(s) followed by a Carriage return (0x0D), Line Feed (0x0A), and a Colon (:). If the command was rejected, the response will be just a question mark (?) and nothing else. If the command is not expected to return a value, the response will be just the Colon (:).

4) Closing the Socket

The socket should be left open during general operation to send/receive commands. In order to close the connection, the programming language should have a “CloseSocket” that will close the TCP/IP connection to the controller.

Notes:

- 1) A single command should not exceed 80 characters, however multiple commands can be grouped together in a single packet by separating them with semicolons.
- 2) Packet size should not exceed 450bytes
- 3) See the Appendix of this document to see example packets of sending commands and receiving responses

Part 2 – Advanced Communication

There are some cases where a second type of communication is necessary and the data from the controller is “unsolicited”. The two types of situations this occurs is:

1. Data Record output from controller (DR command) over UDP. The Data Record is an internal block of information that can be output by the controller at a regular rate via a UDP handle. The structure for this block of data is specified in Chapter 3 of the controller’s User Manual. The size of the packet depends on the model of the controller and the number of axes. A dedicated UDP handle (socket) should be opened to the controller to receive these packets and process them. The DR command is used to specify the update rate and which handle to direct the Data Record packets to. See the Appendix for a sample DataRecord packet.
2. Messages sent from a program executing on the controller. A program such as this:
#A
MG “Hello”
WT100
JP#A
will output the message “Hello” every 100msec. The best way to receive these unsolicited messages is by opening a dedicated socket for them and directing the output via the CF command. There is a data adjustment bit (CW1) that can be set so that these messages have their most significant bit set high. This is a way that can be used to differentiate these messages from a standard command response and is especially useful if the unsolicited messages are sent out the same socket as the command responses (not recommended).
3. EI Event Interrupts (Accelera controllers). The EI command allows the user to generate a UDP interrupt message with a specific status byte based on an interrupt condition. Some examples of interrupt conditions are a motion complete event, limit switch trigger, excess position error, digital input, or a user generated interrupt (UI). The [complete list](#) of events as well as the status byte value is shown in the Command Reference of the controller.

Appendix B: Initiating a Connection from host to controller (3 packets)

The simplest example is when both devices already have a static IP address assigned. In the case shown here, the host has an IP address of 192.168.1.1 and the Galil controller has an IP address of 192.168.1.2. The following 3 packets show the sequence of the host requesting a connection (setting SYN=1), and then the controller acknowledging the request with a SYN=1 and ACK=1, and finally the host responding with an ACK=1. **NOTE:** This is part of the standard TCP/IP protocol – these packets should be generated automatically as part of the “OpenSocket” type of function provided by the programming languages TCP/IP socket interface. They are shown here for reference purposes only.



