Introduction to a JNIOR Protocol Connection

Updated June 18, 2008

This document provides an introduction into the methods and logistics for communicating from an application residing on a personal computer (or other device) with the JNIOR 310 utilizing the JNIOR Protocol over an Ethernet network using TCP. This document is an introduction to the more detailed document describing the JNIOR Protocol. The JNIOR Protocol document describes the various communication packets that are sent back and forth between the JNIOR and an application over the Ethernet to read and write to the JNIOR I/O and other settings.

The JNIOR device acts as a server for the TCP communications. The default setting is for the JNIOR to listen and accept connections coming in over the Ethernet on port 9200. [NOTE: This port is a configurable setting that can be altered via the JNIOR registry. The registry key for this setting is `JniorServer/Port`. The registry key can be changed from the main JNIOR web page and/or via a Telnet window using the Registry editor. These are further described in the JNIOR Web Based Interface Manual and Command Line Communications Manual.] The remote application makes the initial contact with the JNIOR by creating a socket connection to the JNIOR. In order to receive a response from the JNIOR, you must either successfully login or disable login by modifying the appropriate JNIOR Registry setting. After a successful login, the JNIOR automatically sends out the Monitor packet with any changes in the I/O state. The Monitor packet contains the status of all the I/O and is described in the JNIOR Protocol document. The remote application can parse out this packet and get the desired information.

The following is a summary of what a programmer should initially expect when implementing the JNIOR protocol.

1. Establish a socket connection over port 9200 and login with a valid user name and password **(message type 126).** "jnior", "jnior" is the default admin user name and password.
2. Based on the login user name and password, the JNIOR will reply with the login acknowledgement message **(message type 125)** alerting the application of the security level granted.
3. The JNIOR will issue a Monitor packet **(message type 1)**. This monitor packet has information about the JNIOR and the current I/O states.

Once connected and logged in, the application can communicate with the JNIOR by following the JNIOR protocol and issuing any of the commands. The login is necessary before any commands are sent to the JNIOR with the exception of registry reads **(message type 11)**.

Please note, that when you send an I/O command to the JNIOR, a Monitor packet **(message type 1)** does not automatically get returned. The Monitor packet **(message type 1)** only gets returned after I/O commands that actually cause the I/O states to alter. The Monitor packet is automatically issued with each change in I/O state unless the Monitor packet was disabled.

For those requiring finer control of the Ethernet packets, the user application can issue a command to the JNIOR disabling the Monitor packet using the Request packet **(message type 5)** for the current connection (it is not global for all connections, only for this particular connection because multiple connections can be made to the JNIOR at the same time). The user would then issue a command to Subscribe **(message type 15)** to various I/O points. The JNIOR will then notify this particular connection of the change in the subscribed to I/O points.

If you need to continuously interact with the JNIOR I/O, it is better to leave the connection open. However, the JNIOR has a 15 minute timeout such that if it doesn't see any activity on a particular connection, it will close the connection. This is a failsafe so that unused connections do not build on the JNIOR should the Ethernet link be physically broken or the other application stop. In order to keep the connection active, a "keep alive" packet needs to be sent to the JNIOR approximately ever 10 minutes. This is described in the JNIOR Protocol document on page 5 – Connection Maintenance (Keep Alive).

Example Packets

The following example packets were captured using WireShark, formerly Ethereal. WireShark is an exceptional tool for debugging the implementations of communication protocols. This gives you a sample of the format of the packets you will be receiving and sending.

Monitor (message type 1)

```
01 00 60 68 85 01 0e 6a  72 33 31 30 20 76 32 2e ..`h...j r310 v2.
31 34 2e 31 37 00 00 00  00 00 00 00 00 00 00 00 14.17... ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01 ........ ........
19 33 ca 9f eb                                    .3...
```

Login Request Packet (message type 126)

```
01 00 0d 60 b7 7e 05 6a  6e 69 6f 72 05 6a 6e 69 ...`.~.j nior.jni
6f 72                                             or
```

Login Response Packet (message type 125)

```
01 00 02 f0 20 7d 80                              .... }.
```

Read Registry Keys Packet (message type 11)

```
01 00 13 be 61 0b 00 01  00 de 0d 24 53 65 72 69 ....a... ...$Seri
61 6c 4e 75 6d 62 65 72                          alNumber
```

Registry Response Packet (message type 12)

```
01 00 0f 9e d2 0c 00 01  00 de 09 31 30 35 31 30 ........ ...10510
30 33 32 38                                      0328
```