bSerialSplit Driver

bSerialSplit is a driver that can split/duplicate an input serial connection into several (up to 5) output serial connection(s). Data passing through the driver can be displayed as ASCII, Hex and/or Binary formats in the standard Lua Output window. bSerialSplit can also inject data into a serial connection and when paired with bSerialProxy, remote serial devices/drivers can connect to a local serial input.

bSerialSplit was created as a development tool for another driver (bPentair) that is connected to the to-be controlled equipment via direct serial connection. I recall seeing and eventually found Ryan E's SerialSpy driver that allowed you to inspect the passing serial traffic. This was a great start and inspiration but for a development tool, I had a few additional "wants". See below for discussion on Split Serial Connection, Head Pattern, Injection and Remote Serial Connection.

Setup

1) Within your Control4 project, install an instance of the bSerialSplit driver for every serial connection you would like to split, inspect or proxy
2) In Composer Connections, connect bSerialSplit's RS-232 Input and a RS-232 Output so that it is "in the middle" of two serial endpoints. You can connect up to 5 RS-232 Output devices
3) Set the Show Ascii/Hex/Binary properties for the type of data you are expecting. Note that not all Hex and Binary data have an Ascii equivalent
4) Depending on the verbosity of the endpoints, you should immediately start seeing the serial data in the Lua Output. Use Debug Level "3 - Info" for the cleanest data view

Properties

Driver Setup

Upgrade Mode	BNet Solutions drivers can automatically update themselves. Options are "Automatic", and "Upgrade Now". See "Upgrade Mode" section below
Admin Server Enabled	Enables / Disables the driver's Admin Server. See "Admin Server" section below
Admin Port	Listening port assigned to Admin Server
Admin Token	Token required to access the driver's Admin Server

License Setup

License Key	Key used to permanently enable driver functionality
License Status	Display current state of driver's license
Driver Version	Installed driver version
MAC Address	Unique network interface identifier for the Controller

bSerialSplit Setup

Show Ascii	Show/Hide the Ascii representation of serial data
Show Hex	Show/Hide the Hex representation of serial data
Show Binary	Show/Hide the Binary representation of serial data

Websocket Server Enabled	Enables/Disables the Websocket Server
Websocket Port	Websocket Server listening port

Serial Debug

Input Type	Input data type in Send To Input and Send To Output properties
Head Pattern	Header pattern to signify the start of a new packet. Can be a comma-deiminated list or blank. For example “ff ff 00 00” or “#,!”. See “Head Pattern” section below for more detail
Flush Buffer Delay	When Head Pattern defined, this is the delay (in milliseconds) that bSerialSplit will wait after the last fragment is received until it flushes and send the buffer contents. Default is 1000 (1 second)
Send To Input	Data to inject/send to the Input endpoint of the serial connection. See “Injection” section below for more detail
Send To Outputs	Data to inject/send to the Output endpoint(s) of the serial connection. See “Injection” section below for more detail
Line Termination	Character sequence to append to the injected data -CR (Carriage Return) -LF (Line Feed) -CRLF (Carriage Return AND Line Feed)

Debug Settings

Debug Mode	Sets where the driver outputs debug information. See “Troubleshooting” section below for more information
Debug Level	Sets how much debug detail the driver outputs

Actions

Send To Input	Sends the data entered in Send To Input property. Used to repeatedly send data after the property is set
Send To Outputs	Sends the data entered in Send To Outputs property. Used to repeatedly send data after the property is set

Split Serial Connection

I’m lucky enough to have a pool, but I don’t have two! I wanted to develop bPentair side-by-side with the existing driver and not continually connect/reconnect the serial connection physically or within Control4. With up to 5 outputs per serial input, I was able to develop sharing the same/only pool input stream.

```
                                  /-RS232 Out1-> bPentair
Pool <-RS232 In-> bSerialSplit <--RS232 Out2-> Pentair Intellitouch
                                  \-RS232 Out3-> SerialLogger
```

Head Pattern

Some serial data is received in nice distinct packets but most of the time this is not the case. Reading and decoding a serial stream gets more tedious and time consuming when packet fragments are split across multiple “reads”. Using the Head Pattern property, bSerialSplit will buffer and display a packet in a single chunk instead of across a series of smaller chunks. Multiple Head Patterns can be entered in a comma-separated list (no spaces unless a space is part of the pattern).

Below is a simple example of a data stream with and without a Head Pattern:

```
Head Pattern: None                              Head Pattern: "#,!00"
Input <-- Pentair    Ascii :  #POOLHT?          Input <-- Pentair    Ascii :  #POOLHT?
Input --> Output(s)  Ascii :  !00               Input --> Output(s)  Ascii :  !00 POOLHT=0
Input --> Output(s)  Ascii :  POOLHT =
Input --> Output(s)  Ascii :  0

Input <-- Pentair    Ascii :  #SPAHT?           Input <-- Pentair    Ascii :  #SPAHT?
Input --> Output(s)  Ascii :  !                 Input --> Output(s)  Ascii :  !00 SPAHT=1
Input --> Output(s)  Ascii :  00
Input --> Output(s)  Ascii :  SPAHT=
Input --> Output(s)  Ascii :  1
```

When Head Pattern is blank, bSerialSplit will display and forward data as it is received: 1 write for 1 read. When Head Pattern is defined, bSerialSplit will flush (and write) the buffer after Flush Buffer Delay. This is necessary when the input data stream is sparse or irregular.

INJECTION

bSerialSplit can inject data into the serial connection using the Send To Input and Send To Outputs properties. Set the Input Type property (Ascii, Hex or Binary) to the appropriate data type and enter the data to send in the appropriate input/output property. The data will be sent when the property is "Set". For repeat Sends (after the property is set), use the Send To Input and Send To Outputs Actions.

Set the Line Termination property to the appropriate line termination sequence. Options are LF (Line Feed), CR (Carriage Return), CRLF (Carriage Return and Line Feed) and None. The below examples use CR Line Termination (Hex code 0d and binary 00001101).

When sending to input, "Output(s)" is replaced by Debug in the Lua Output and the Input Type is marked with a ">" to show what was actually sent. Note that the data is sent to the connected C4 Control Input and NOT to any other connected C4 Control Outputs. For example:

```
Input <-- Debug      Ascii : >Test
Input <-- Debug      Hex   :  54 65 73 74 0d
Input <-- Debug      Binary:  01010100 01100101 01110011 01110100 00001101
```

When input is injected from a connected bSerialProxy, "Output(s)" is replaced by "WS User":

```
Input <-- WS User    Ascii : >Test
Input <-- WS User    Hex   :  54 65 73 74 0d
Input <-- WS User    Binary:  01010100 01100101 01110011 01110100 00001101
```

When sending to outputs, "Input" is replaced by "Debug" in the Lua Output and the Input Type is marked with a ">" to show what was actually sent. Note that the data is sent to all connected C4 Control Outputs. For example:

```
Debug --> Output(s)  Ascii : >Test
Debug --> Output(s)  Hex   :  54 65 73 74 0d
Debug --> Output(s)  Binary:  01010100 01100101 01110011 01110100 00001101
```

REMOTE SERIAL CONNECTION

My development controller is separate hardware from my production controller and the two are not in close proximity. By connecting a bSerialProxy* instance to bSerialSplit's Websocket

Server, I am able to develop on my development server but use the pool's input serial connection from my production controller. The setup looks like this:

```
 [-        Production Env      -]    [-         Development Env          -]
Pool <-RS232 In-> bSerialSplit <-WS-> bSerialProxy <-RS232 Out-> bPentair
```

*bSerialProxy is a separate driver from bSerialSplit and can be found on the BNet Solutions website.

Development Driver

bSerialSplit and bSerialProxy are intended for use as development tools. While I have run these drivers on my production controller since early 2020, the volume of data over some serial connections, data transformations (read: lots of strings, bit operations and "math" that Lua is not known to excel at) and the extra serial and socket I/O all equals more "work".

Admin Server

BNet Solutions drivers' have a built-in webserver that looks and behaves like the Properties, Actions and Lua Output tabs for the driver in Composer. The Admin Server's default port for the bSerialSplit driver is 41500 and is configurable in the driver's properties. Using a web browser, navigate to http://[controller ip]:41500 where "controller ip" is the IP Address of your Control4 Director (EA5, EA3, EA1, etc). For example, http://192.168.1.100:41500. The Admin Server is protected by a challenge page that requires a token to continue. By default, the token is "bSerialSplitAdmin". Once authenticated, the token is stored in a cookie (technically hashed, then stored) so you won't need to log in every time. The token is configurable via the driver's property page. The Admin Server is enabled by default but can be disabled entirely via the driver's property page.

Upgrade Mode

BNet Solutions drivers can automatically update themselves. New driver functionality or capability is typically packaged as an incremental "Major" version (v3, v4 etc). "Minor" versions (v2.3, v2.4) are typically maintenance releases that update underlying libraries, address a specific issue or usability concern.

Automatic	When "Automatic" is selected, the driver will upgrade/update itself when a new version is available. This is currently the only option.
Update Now	Checks for and upgrades to any newer Major or Minor version.

Known Issues and Limitations

- Requires C4 OS v2.10.X or greater
- C4 defines serial connections on a per-driver basis, not per-input/output connection. This driver is expecting 9600 baud, no parity bit and 1 bit stop (9600/8-N-1). Other serial settings will likely not work properly

Troubleshooting

All BNet Solutions products have an additional 'Submit' Debug Mode. With this mode selected, the driver creates a unique log file to capture the Lua output based on the selected Debug Level (usually set to "5 - Debug"). Once 'Submit' Debug Mode is deselected, either manually or when

the Debug Timer expires, the Submit Debug Log is uploaded to the BNet Solutions Server for analysis.

The server notifies me when Submit files are uploaded but if you have not purchased a license, I will have no way to reach back out to you for troubleshooting so please email me your contact information.

Legal

By using this driver, you are indicating that you have read and agree with the Policies and Terms that govern its usage as published here.

My Contact Information

You can reach me at blucas@bnet4solutions.com for comments or questions.

Change Log

v1 - 08/21 Initial Release