1) Castle Serial Link Overview

The Serial Link device will allow customers to communicate with Castle ESCs through the Castle Link Live protocol, which allows access to real time telemetry feedback from the ESC. The device is capable of communicating through several serial protocols (e.g. TTL Serial, I2C, SPI). The serial protocols allow the user to control the connected ESC’s throttle level in real time while reading current operating conditions such as battery voltage and motor RPM. The device also has the ability to be controlled through an analog input, or can be used in a pass-through mode. The pass-through modes allow the Serial Link to receive the throttle signal from either an Analog or PPM input, and still allow a TTL Serial or I2C connection to pull real time data from the ESC.

For more information about the Castle Link Live Protocol see: www.castlecreations.com/CastleLinkLive

For the most up to date version of this document see: www.castlecreations.com/CastleSerialLink

2) Device Pin-out

(a) - Castle Serial Link Pin-out

3) Pin Definitions

Referencing image (a) above, the following table describes the purpose for each pin.

<table>
<thead>
<tr>
<th>Connection</th>
<th>TTL Serial</th>
<th>SPI</th>
<th>I2C</th>
<th>Analog</th>
<th>PPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX Port</td>
<td>-GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX Port</td>
<td>+BUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX Port</td>
<td>Signal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>+BUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>+5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>A</td>
<td>TX</td>
<td>CLK</td>
<td>SDA</td>
<td>Analog I/O A</td>
</tr>
<tr>
<td>Connector</td>
<td>B</td>
<td>RX</td>
<td>MISO</td>
<td>SCL</td>
<td>Analog I/O B</td>
</tr>
<tr>
<td>Connector</td>
<td>C</td>
<td>n/a</td>
<td>MOSI</td>
<td>n/a</td>
<td>Analog I/O C</td>
</tr>
<tr>
<td>Connector</td>
<td>D</td>
<td>n/a</td>
<td>NSS</td>
<td>n/a</td>
<td>Analog I/O D*</td>
</tr>
<tr>
<td>Connector</td>
<td>-GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Analog Channel D is recommended because it includes a small filter capacitor.

(b) - Castle Serial Link Pin Definitions
4) Powering the Serial Link

The Serial Link needs to be powered through the +BUS connection. ESC’s with internal BEC’s will power the device automatically, but an external supply will need to be attached if the connected ESC does not contain a BEC (See figure (c) below for a wiring diagram).

5) Adjusting Settings

A Castle Link device can be used to change the settings of the Serial Link device. The Serial Link can be connected directly to a Castle Link device with the included wire adapter.

6) TTL Serial Wiring Diagram

A Generic FTDI USB to Serial converter can be used to connect the Serial Link to a PC. The Serial Link will communicate with either a 3.3V device or a 5.0V device. When wiring the devices make sure to cross the TX and RX signals, if this is not done the device will not communicate.
7) Communication Specifications

The following table shows the available communication specifications and settings which can be configured using a Castle Link device.

- **Output Modes**
  - Castle Link Live Protocol
    - Real Time Telemetry Feedback
    - 1.0 ms to 2.0 ms throttle signal
    - 100 Hz throttle refresh rate
  - PPM (Hobby Signal)
    - 1.0 ms to 2.0 ms throttle signal
    - 50, 100, 200, or 400 Hz throttle refresh rate

- **Input Modes**
  - TTL Serial
    - Device ID (0 to 63)
    - Baud Rate (1200 to 230400 baud)
  - I2C
    - 7bit I2C Slave Address (8 to 71)
    - I2C Frequency (10 kHz to 400 kHz)
  - SPI
    - Device ID (0 to 63)
    - SPI Frequency (125 kHz to 500 kHz)
  - Analog
    - Port (A, B, C, or D)
    - Range
      - Normal (0V to 5V -> 1.0ms to 2.0ms throttle)
      - Inverted (0V to 5V -> 2.0ms to 1.0ms throttle)
      - Lower Half (0V to 5V -> 1.0ms to 1.5ms throttle)
      - Upper Half (0V to 5V -> 1.5ms to 2.0ms throttle)
      - Lower Half Inverted (0V to 5V -> 1.5ms to 1.0ms throttle)
      - Upper Half Inverted (0V to 5V -> 2.0ms to 1.5ms throttle)

- **Pass-Through Modes**
  - TTL Serial (with Analog Input)
    - TTL Serial for real time telemetry feedback
    - Analog for throttle control
  - I2C (with Analog input)
    - I2C for real time telemetry feedback
    - Analog for throttle control
  - TTL Serial (with PPM Input)
    - TTL Serial for real time telemetry feedback
    - PPM for throttle control
  - I2C (with PPM Input)
    - I2C for real time telemetry feedback
    - PPM for throttle control

* The pass-through modes allow the user to control the throttle level using a receiver or manual control using a potentiometer. These modes will allow the user to read the real time telemetry feedback from the ESC if Link Live mode is enabled.
8) Register Description

The three digital forms of communication (TTL Serial, SPI, and I²C) are implemented by reading / writing to a set of 16-bit registers. The available registers are described in the tables below. Note: that the registers are divided into Read and Write registers.

- **Read Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Voltage</td>
<td>The controller’s input voltage</td>
</tr>
<tr>
<td>1</td>
<td>Ripple</td>
<td>The controller’s input voltage ripple</td>
</tr>
<tr>
<td>2</td>
<td>Current</td>
<td>The controller’s current draw</td>
</tr>
<tr>
<td>3</td>
<td>Throttle</td>
<td>The controller’s commanded throttle value</td>
</tr>
<tr>
<td>4</td>
<td>Power</td>
<td>The controller’s output throttle percentage</td>
</tr>
<tr>
<td>5</td>
<td>Speed</td>
<td>The motors electrical RPM</td>
</tr>
<tr>
<td>6</td>
<td>Temp</td>
<td>The controller’s temperature</td>
</tr>
<tr>
<td>7</td>
<td>BEC Volt</td>
<td>The BEC’s voltage</td>
</tr>
<tr>
<td>8</td>
<td>BEC Current</td>
<td>The BEC’s current load</td>
</tr>
<tr>
<td>9</td>
<td>Raw NTC</td>
<td>The raw NTC temperature value</td>
</tr>
<tr>
<td>10</td>
<td>Raw Linear</td>
<td>The raw linear temperature value</td>
</tr>
<tr>
<td>25</td>
<td>Link Live</td>
<td>Whether the Serial Link is in Link Live mode</td>
</tr>
<tr>
<td>26</td>
<td>Fail Safe</td>
<td>The E. Stop/RX Glitch fail safe output (0 = 1ms; 100 = 2ms)</td>
</tr>
<tr>
<td>27</td>
<td>E. Stop</td>
<td>If ‘1’ output is set to fail safe output</td>
</tr>
<tr>
<td>28</td>
<td>Packet In</td>
<td>The number of packets received by the serial link</td>
</tr>
<tr>
<td>29</td>
<td>Packet Out</td>
<td>The number of packets sent by the serial link</td>
</tr>
<tr>
<td>30</td>
<td>Check Bad</td>
<td>The number of received packets with invalid checksums</td>
</tr>
<tr>
<td>31</td>
<td>Packet Bad</td>
<td>The number of received packets with invalid data</td>
</tr>
</tbody>
</table>

(f) - Read Register Descriptions

- **Write Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
<th>Description</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Throttle</td>
<td>The controller’s commanded throttle value</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>129</td>
<td>Fail Safe</td>
<td>The E. Stop/RX Glitch fail safe output (0 = 1ms; 100 = 2ms)</td>
<td>0 to 100</td>
</tr>
<tr>
<td>130</td>
<td>E. Stop</td>
<td>If ‘1’ output is set to fail safe output</td>
<td>0 or 1</td>
</tr>
<tr>
<td>131</td>
<td>Packet In</td>
<td>The number of packets received by the serial link</td>
<td>Sets to 0</td>
</tr>
<tr>
<td>132</td>
<td>Packet Out</td>
<td>The number of packets sent by the serial link</td>
<td>Sets to 0</td>
</tr>
<tr>
<td>133</td>
<td>Check Bad</td>
<td>The number of received packets with invalid checksums</td>
<td>Sets to 0</td>
</tr>
<tr>
<td>134</td>
<td>Packet Bad</td>
<td>The number of received packets with invalid data</td>
<td>Sets to 0</td>
</tr>
</tbody>
</table>

(g) - Write Register Descriptions

* Unlisted register addresses are reserved for future use
9) Throttle Register

The ESC can be controlled by writing to the throttle register. The response of the speed control to writes to this register depends on the ESC current settings (adjustable via Castle Link). The Serial Link controls the ESC in essentially the same manner that a receiver does. Writing a 0 [0x0000] to this register would be the same as the receiver sending a 1.0ms pulse, OFF. Writing 65535 [0xFFFF] to this register would be the same as the receiver sending a 2.0ms pulse, FULL THROTTLE.

In most applications, a Fixed Throttle mode is suggested. This will result in a consistent ESC response to register values. However, all throttle modes work, including Airplane Auto-Calibration and the Governor modes. If a governed RPM is desired, governor-low or governor-high mode is suggested.

When the Serial Link is set to one of the pass through combination modes (e.g. TTL Serial (with Analog), I²C (with PPM)...) the communication protocol will not have control over the throttle register. Note that it is possible to set the Emergency Stop register which will set the throttle output to the Fail Safe register’s value.

10) Safety Features

The serial link device has multiple safety features which can be used, the features are outlined below.

- **Emergency Stop**

  The Serial Link has a built-in emergency stop override which can be set in all modes. If the E. Stop register is set to ‘1’ then the Serial Link will ignore the throttle register and transmit the contents of the Fail Safe register. The Fail Safe register can be set to values between 0 and 100 inclusive, where 0 represents 1.0ms and 100 represents 2.0ms. The E. Stop feature was added so that the Analog and PPM combination modes could be disabled by the TTL Serial /I²C protocol. The default value of the Fail Safe register is 1.0ms. The value should be set to 50 (1.5ms) if reversible/car esc’s are used with the Serial Link. The Red LED will turn on when the Fail Safe output is active.

- **Throttle Glitch Detection**

  If the Serial Link is in one of the PPM combination modes and the PPM throttle signal is lost for more than 1 second then the Serial Link will go into the throttle glitch state. While in the throttle glitch state the Serial Link will output the Fail Safe registers value. The Red LED will turn on when in the throttle glitch state. Once the input returns the throttle output will change back to the PPM input’s value and the Red LED will turn off.

- **Communication Watchdog Timer**

  The Communication Watchdog Timer is a safety feature that will set the throttle output to the Fail Safe value if no communication packets are received in a programmable number of seconds. This value can be set to 0 to disable the Communication Watchdog Timer, or can be set from 1 to 255 seconds. The Red LED will turn on when the Fail Safe output is active.
11) Conversion

The following table lists the conversion factors necessary to turn the register values into an actual value.

<table>
<thead>
<tr>
<th>Data Item</th>
<th>Scale</th>
<th>Units</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>20.0</td>
<td>Volts</td>
<td>100</td>
</tr>
<tr>
<td>Ripple Voltage</td>
<td>4.0</td>
<td>Volts</td>
<td>20</td>
</tr>
<tr>
<td>Current</td>
<td>50.0</td>
<td>Amps</td>
<td>250</td>
</tr>
<tr>
<td>Throttle</td>
<td>1.0</td>
<td>Milliseconds</td>
<td>2.5</td>
</tr>
<tr>
<td>Output Power</td>
<td>0.2502</td>
<td>Percent</td>
<td>1</td>
</tr>
<tr>
<td>RPM</td>
<td>20,416.66</td>
<td>Electrical RPM</td>
<td>100,000</td>
</tr>
<tr>
<td>BEC Voltage</td>
<td>4.0</td>
<td>Volts</td>
<td>20</td>
</tr>
<tr>
<td>BEC Current</td>
<td>4.0</td>
<td>Amps</td>
<td>20</td>
</tr>
<tr>
<td>Temperature</td>
<td>30.0</td>
<td>Degrees C</td>
<td>150</td>
</tr>
<tr>
<td>Raw NTC Temperature*</td>
<td>63.8125</td>
<td>Units**</td>
<td>255</td>
</tr>
<tr>
<td>Raw Linear Temperature*</td>
<td>30.0</td>
<td>Degrees C</td>
<td>150</td>
</tr>
</tbody>
</table>

(g) - Conversion Factors

The value is computed by the following equation:

\[
\text{Result} = \frac{\text{Register}}{2042} \times \text{Scale}
\]

For example, if the Voltage register holds a value of 4084, the actual voltage is:

\[
\text{Voltage} = \frac{4084}{2042} \times 20 = 40.0V
\]

* The Serial Link calculates the degrees Celsius value for you using a lookup table, but if more accuracy is needed it is possible to read the raw temperature sensor values from the esc. Note: Castle esc’s contain either a Linear or NTC thermistor but never both. In order to detect which temperature sensor is used compare the Raw NTC and the Raw Linear, the register with the larger value is used.

** It is possible to read the raw output from the NTC thermistor, but this requires additional processing to find the degrees Celsius value. Use the formula below to find the temperature in Celsius.

\[
\text{Degrees } C = \left( \frac{1}{\ln(\text{value} \times R2 / (255 - \text{value}) / R0 + 1 / 298)} \right) - 273
\]

where \( R0 = 1000; R2 = 10200; B = 3455 \)
12) Communication Protocols

- TTL Serial

TTL Serial communication with the Serial Link takes the form of reading and writing to a set of 16-bit registers. Every TTL Serial command is 5 bytes long and every response is 3 bytes long. See figure (g).

The first byte of the command contains a start bit and the Device ID of the Serial Link to communicate with. Device ID’s can range from 0 to 63, allowing multiple Serial Links on the same bus. The second byte specifies the register number; note that there are separate addresses for reading and writing. The remaining bytes contain the data to write to the register, the Command Data is ignored if it is a read address, and a checksum to ensure that the transfer was not corrupted.

The three byte response will return the value of the register specified by the command. If the command pointed to an invalid register or the command was corrupted, 0xFFFF will be returned to indicate an error. The response will also include a checksum for the Response Data so that it can be verified.

At any time you can write a series of at least 5 0x00 bytes to clear the command buffer. This is generally a good idea upon initialization to ensure that the controller is in synch with the ESC.

The checksum is a modular sum. Correctly compute it as follows:

\[ \text{Checksum} = 0 - (\text{Byte 0} + \text{Byte 1} + \text{Byte 2} + \text{Byte 3}) \]

If the checksum is correct, the result of adding the bytes in the command or response packet together will be 0x00 (ignoring overflows). The response checksum can be verified by adding the Response Data bytes and the response checksum, if valid they will total to 0x00 (ignoring overflows).
SPI communication with the Serial Link takes the form of reading and writing to a set of 16-bit registers. Every SPI command is 5 bytes long and every response is 3 bytes long. See figure (h).

The first byte of the command contains a start bit and the Device ID of the Serial Link to communicate with. Device ID’s can range from 0 to 63, allowing multiple Serial Links on the same bus. The second byte specifies the register number; note that there are separate addresses for reading and writing. The remaining bytes contain the data to write to the register, the Command Data is ignored if it is a read address, and a checksum to ensure that the transfer was not corrupted.

Since SPI is synchronous the master will have to write at least 5 bytes in order to receive the three byte response which will return the value of the register specified by the command. If the command pointed to an invalid register or the command was corrupted, 0xFFFF will be returned to indicate an error. The easiest way to accomplish reading and writing in SPI is to use the next transmission packet to receive the previous packets response. The response will also include a checksum for the Response Data so that it can be verified.

At any time you can write a series of at least 5 0x00 bytes to clear the command buffer. This is generally a good idea upon initialization to ensure that the controller is in synch with the ESC.

The checksum is a modular sum. Correctly compute it as follows:

\[
\text{Checksum} = 0 - (\text{Byte 0} + \text{Byte 1} + \text{Byte 2} + \text{Byte 3})
\]

If the checksum is correct, the result of adding the bytes in the command or response packet together will be 0x00 (ignoring overflows). The response checksum can be verified by adding the Response Data bytes and the response checksum, if valid they will total to 0x00 (ignoring overflows).
- **I^2C**

I^2C communication with the Serial Link takes the form of reading and writing to a set of 16-bit registers. Every I^2C command is 5 bytes long and every response is 3 bytes long. See figure (i).

The first byte of the write command specifies the 7 bit I^2C Slave Address of the Serial Link to communicate with. Valid Addresses can range from 8 to 71, allowing multiple Serial Links on the same bus. The second byte specifies the register number and the remaining bytes contain the data to write to the register, note that there are separate addresses for reading and writing. The last byte is a checksum to ensure that the transfer was not corrupted.

The response will need to be requested by sending the 7 bit I^2C Slave Address with a 1 to indicate a Read command. The response will then return the value of the register specified by the command. If the command pointed to an invalid register or the command was corrupted, 0xFFFF will be returned to indicate an error. The response will also include a checksum for the Response Data so that it can be verified.

The checksum is a modular sum. Correctly compute it as follows:

\[
\text{Checksum} = 0 - (\text{Byte 0} + \text{Byte 1} + \text{Byte 2} + \text{Byte 3})
\]

If the checksum is correct, the result of adding the bytes in the command or response packet together will be 0x00 (ignoring overflows). The response checksum can be verified by adding the Response Data bytes and the response checksum, if valid they will total to 0x00 (ignoring overflows).

(j) - I^2C Command / Response Protocol
13) Additional Communication Modes

- **Analog Input**

The Serial Link also has the capability to control the esc’s throttle using an Analog input on one of the inputs. The available channels are labeled A, B, C, and D, channel D is recommended because it includes a small filter capacitor. Voltages from 0V to 5V are accepted and will be converted to a throttle output depending on the Analog Range Select setting, see figure (j).

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>0V to 5V -&gt; 0% to 100% throttle</td>
</tr>
<tr>
<td>Inverted</td>
<td>0V to 5V -&gt; 100% to 0% throttle</td>
</tr>
<tr>
<td>Lower Half</td>
<td>0V to 5V -&gt; 0% to 50% throttle</td>
</tr>
<tr>
<td>Upper Half</td>
<td>0V to 5V -&gt; 50% to 100% throttle</td>
</tr>
<tr>
<td>Inverted Lower Half</td>
<td>0V to 5V -&gt; 50% to 0% throttle</td>
</tr>
<tr>
<td>Inverted Upper Half</td>
<td>0V to 5V -&gt; 100% to 50% throttle</td>
</tr>
</tbody>
</table>

(k) - Analog Range Options

- **TTL Serial/I\(^2\)C (with Analog Input)**

These combination modes allow external control through an Analog input, only channel D can be used, and also allow real time data feedback using either TTL Serial or I\(^2\)C. Note that when in these modes, the communication protocol does not have write access to the Throttle register. If needed an Emergency Stop register was added which can be set by the communication protocol. If set the Emergency Stop register will override the Analog input and use the value set in the Fail Safe register.

Look at the headings above for details into how the individual modes work.

- **TTL Serial/I\(^2\)C (with PPM Input)**

These combination modes allow external control through a PPM input on channel D and also allow real time data feedback using either TTL Serial or I\(^2\)C. Note that when in these modes, the communication protocol does not have write access to the Throttle register. If needed an Emergency Stop register was added which can be set by the communication protocol. If set the Emergency Stop register will override the Analog input and use the value set in the Fail Safe register.

The PPM input modes allow the Serial Link to pass through the incoming signal from a receiver and then output the same signal to the esc but with Link Live communication enabled. This allows an external device to pull real time data from a standard human controlled receiver/esc pair.

The PPM data accepts values from 1ms to 2ms anything above or below these thresholds will be clipped. If at any point the PPM input quits for more than 1 second the Serial Link will go into the RX Glitch State and will output the throttle value in the Fail Safe register.

Look at the headings above for details into how the individual modes work.
Castle Serial Link – Communication Protocol

Subject to change at any time without notice or warning.

Revision Log:

1.0 – 19-Oct-2010:
   Initial Version

1.1 – 12-Aug-2011:
   Changed register numbers for Packet Count registers

1.2 – 24-Jan-2013:
   Updated/restructured to support multiple communication modes

1.3 – 16-Apr-2013:
   Updated for production release

1.4 – 17-Jun-2015:
   Changed RS232 to TTL Serial

1.5 – 23-Dec-2015:
   Added a section on FTDI USB to Serial device wiring