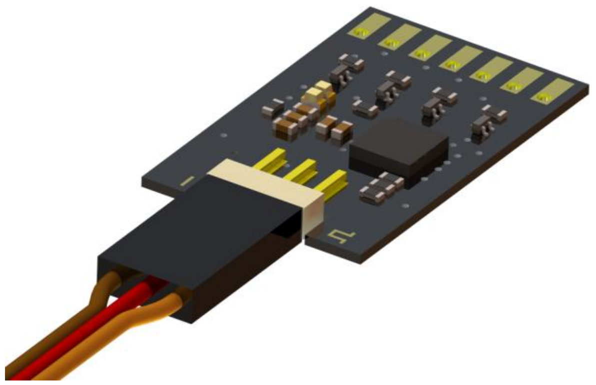


Castle Serial Link – Communication Protocol

Castle Creations, INC.

16-Apr-2013

Version 1.3

castlecreations.com

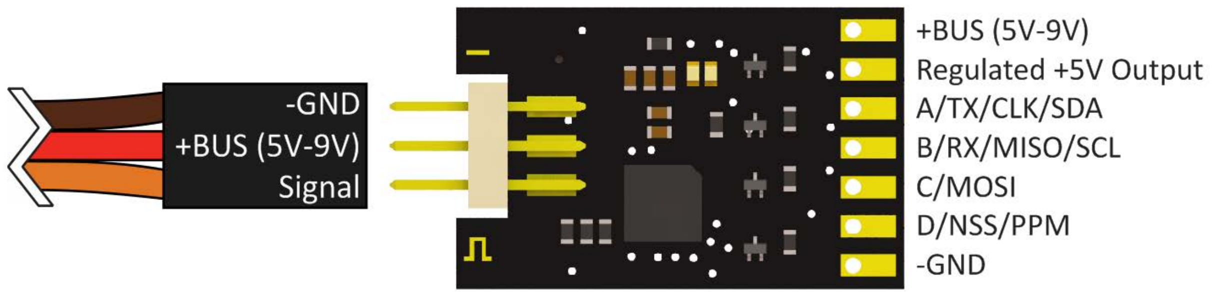
1) Castle Serial Link Overview

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. RS232, 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 an RS232 or I2C connection to pull real time data from the ESC.

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

For the most up to date version of this document see: **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.

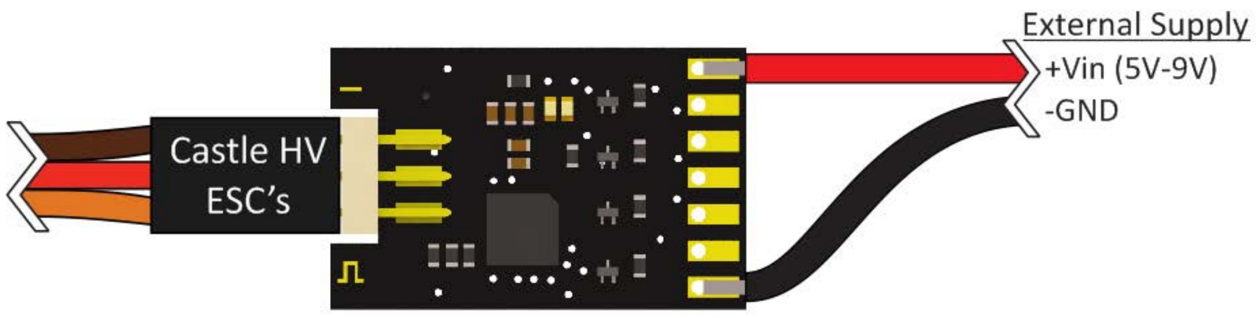
Connection		RS232	SPI	I^2C	Analog	PPM
RX Port	-GND	Controller ground, tied internally to connector ground				
RX Port	+BUS	Power from controller's internal BEC (if present).				
RX Port	Signal	Controller's signal line				
Connector	+BUS	Unregulated Power Bus				
Connector	+5V	Regulated +5.0V supply				
Connector	A	TX	CLK	SDA	Analog I/O A	
Connector	B	RX	MISO	SCL	Analog I/O B	
Connector	C	n/a	MOSI	n/a	Analog I/O C	
Connector	D	n/a	NSS	n/a	Analog I/O D*	PPM Input
Connector	-GND	Ground				

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

(b) - Castle Serial Link Pin Definitions

4) Powering the Serial Link

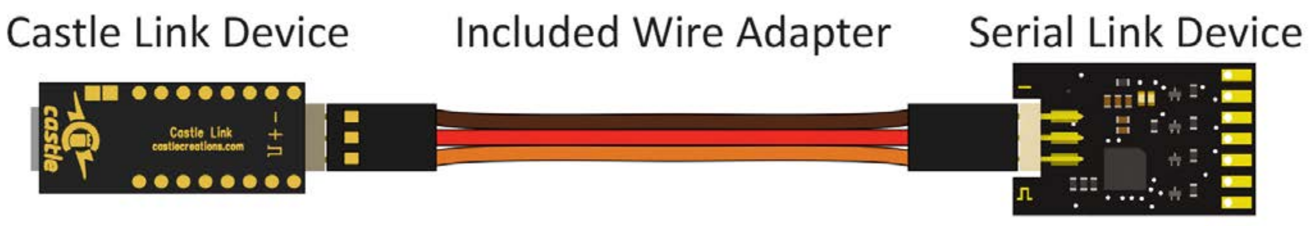
The Serial Link needs to be powered through the +BUS connection. ESC's with internal BEC's will power the device automatically, but an external supply will need to be attached if the connected ESC does not contain a BEC (See figure (c) below for a wiring diagram).



(c) - Connecting a Castle HV ESC

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.



(d) - Connecting the Serial Link and Castle Link

6) Example Applications

- Computer control with telemetry feedback of a Castle ESC
- Embedded control with telemetry feedback of a Castle ESC
- Wireless* control with telemetry feedback of a Castle ESC
- Analog control of a Castle ESC
- Pass through operation to an external data logging device
- Multiple ESC control with telemetry feedback for a Quad/Hex-copter using a single bus and multiple serial link devices

  *External wireless module required

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**
    - RS232
        - 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***
    - RS232 (with Analog Input)
        - RS232 for real time telemetry feedback
        - Analog for throttle control
    - I2C (with Analog input)
        - I2C for real time telemetry feedback
        - Analog for throttle control
    - RS232 (with PPM Input)
        - RS232 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 (RS232, SPI, and I^2C) 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**

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

(e) - Read Register Descriptions

- **Write Registers**

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

(f) - 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 to this register would be the same as the receiver sending a 1.0ms pulse, OFF. Writing 65535 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. RS232 (with Analog), I^2C (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 RS232/I^2C 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.

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

(f) - Conversion Factors

The value is computed by the following equation:

```
Result = Register / 2042 * Scale
```

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

```
Voltage = 4084 / 2042 * 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.

*Degrees C = (1 / (ln(value * R2 / (255 - value) / R0) / B + 1 / 298)) - 273*
where R0 = 1000; R2 = 10200; B = 3455

12) Communication Protocols

- **RS232 (TTL 5V Compliant)**

RS232 communication with the Serial Link takes the form of reading and writing to a set of 16-bit registers. Every RS232 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.

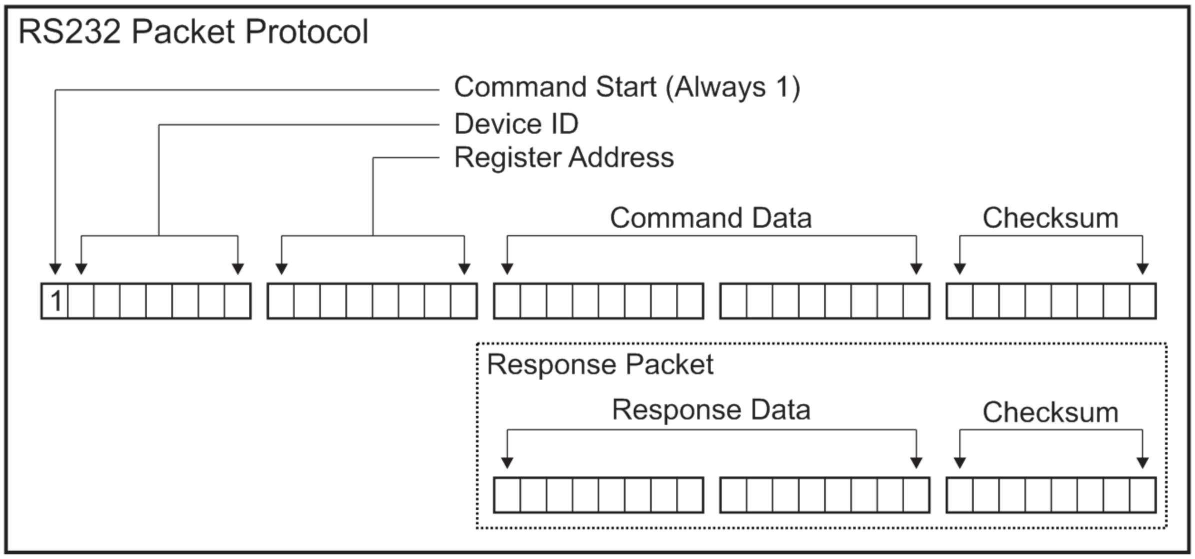
The three byte response will return the value of the register specified by the command. If the command pointed to an invalid register or the command was corrupted, 0xFFFF will be returned to indicate an error. The response will also include a checksum for the Response Data so that it can be verified.

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:

```
  Checksum = 0 - (Byte 0 + Byte 1 + Byte 2 + 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).



(g) - RS232 Command / Response Protocol

- **SPI**

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.

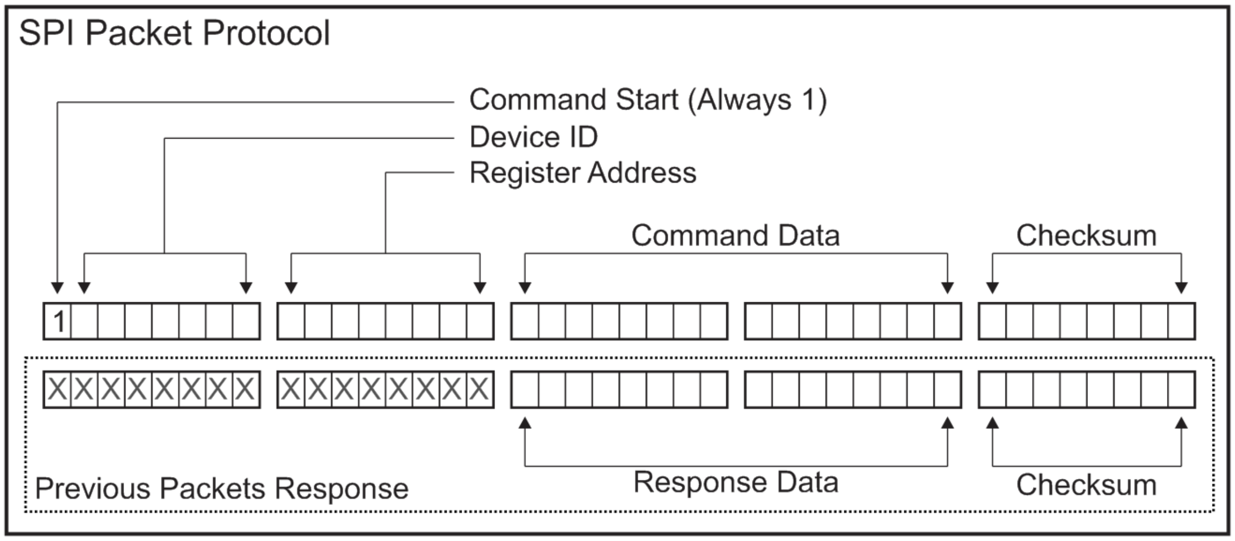
Since SPI is synchronous the master will have to write at least 5 bytes in order to receive the three byte response which will return the value of the register specified by the command. If the command pointed to an invalid register or the command was corrupted, 0xFFFF will be returned to indicate an 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:

```
  Checksum = 0 - (Byte 0 + Byte 1 + Byte 2 + 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).



(h) - SPI Command / Response Protocol

- **I^{2}C**

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

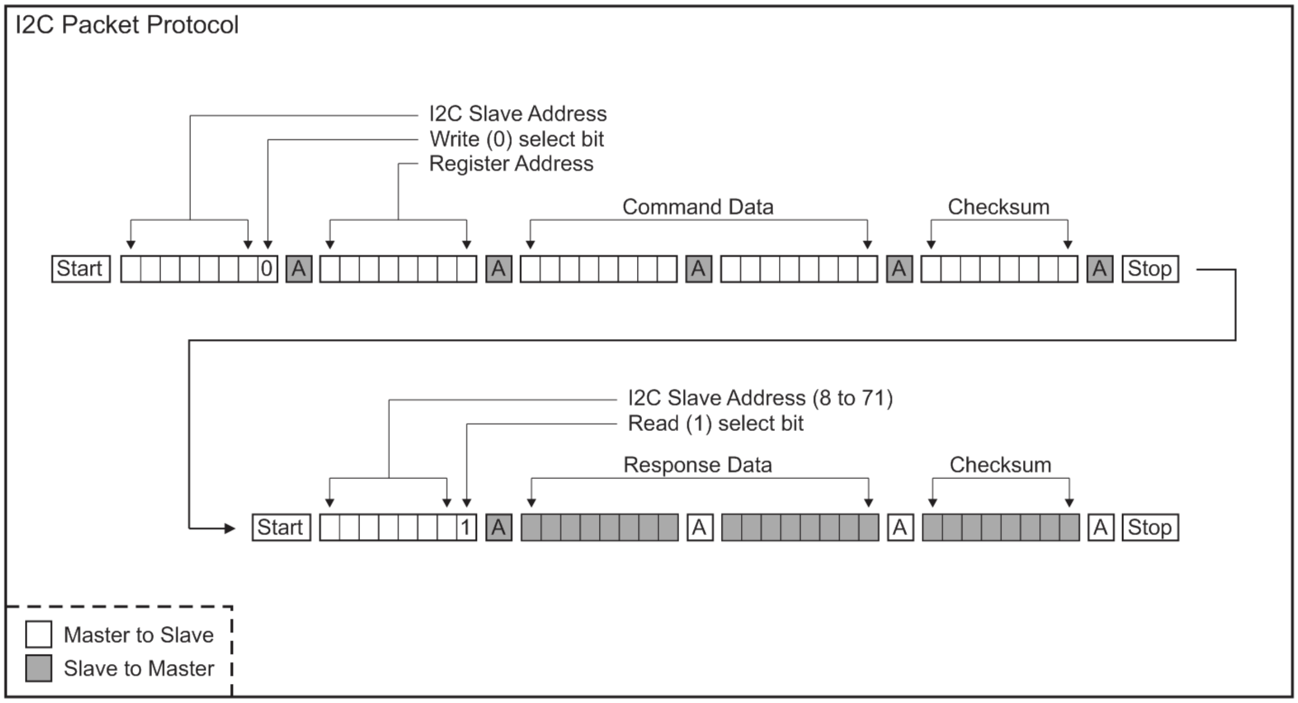
  The first byte of the write command specifies the 7 bit I^{2}C Slave Address of the Serial Link to communicate with. Valid Addresses can range from 8 to 71, allowing multiple Serial Links on the same bus. The second byte specifies the register number and the remaining bytes contain the data to write 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^{2}C 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:

  ```
    Checksum = 0 - (Byte 0 + Byte 1 + Byte 2 + 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) - I^{2}C 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).

Type	Range
Normal	0V to 5V -> 0% to 100% throttle
Inverted	0V to 5V -> 100% to 0% throttle
Lower Half	0V to 5V -> 0% to 50% throttle
Upper Half	0V to 5V -> 50% to 100% throttle
Inverted Lower Half	0V to 5V -> 50% to 0% throttle
Inverted Upper Half	0V to 5V -> 100% to 50% throttle

(j) - Analog Range Options

- **RS232/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 RS232 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.

- **RS232/I2C (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 RS232 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.