HORIZON HOBBY, LLC.

Specification for Spektrum® SRXL2

Based on the Multiplex v2.9

Rev J

June 24, 2019

Contents

Specification for Spektrum® SRXL2

1 INTRODUCTION

This document is intended to serve as a Spektrum-specific guide to describe a bi-directional implementation of the SRXL protocol called SRXL2.

2 LEGAL INFO

This protocol, as described, in its entirety, without warranty or guarantee, can be freely and indefinitely, without limitation, implemented.

3 AUDIENCE

This document is intended for engineers to develop accessories that leverage the SRXL2 protocol. It assumes an intimate knowledge of how the Spektrum systems work.

4 RELATED DOCUMENTATION

All necessary technical information is contained within this document, including diagrams and source code guidance.

Multiplex SRXL specification version 2.9 date 12 Feb 2014 by Walter Meyer.

Specification for Forward Programming, and

Specification for Spektrum Remote Receiver Interface

Telemetry Sensor Struct .h file
https://github.com/SpektrumFPV/SpektrumDocumentation/blob/master/Telemetry/spektrumTelemetrySensors.h

5 ELECTRICAL DATA

The protocol runs on servo-bus power (3.5V-8.4V). This allows us to easily extend the protocol into the servos themselves at a future date.

Signal is standard 3.3V logic with normal UART levels (3.3V = idle line), however the SRXL2 bus is generally in high-impedance state using the micro's internal weak pull-up. This is necessary for idle line detection, although it could leave us open to noise.

Devices on the SRXL2 bus may be powered by the bus, providing they are low-current. It would be unwise to run high-power servos on the same small-gauge wires.

SRXL2 commonly uses Standard 3-pin Servo connectors for connecting to Receivers, in which case the signal line may be used for a standard PWM signal if an SRXL2 device is not found during handshaking. When connecting to SRXL2 Remote Receivers, the ZH4 connector is typically used instead. The pinouts for both are given below.

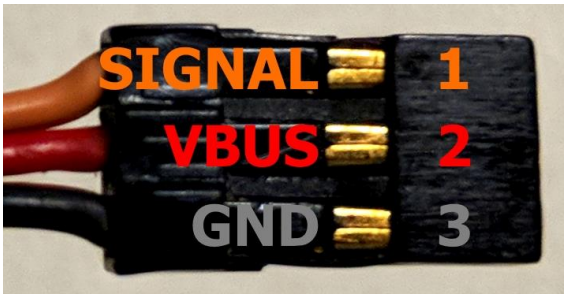
WARNING: Previous SRXL products (e.g. SPM4649T) used the ZH4 connector with a different, non-compatible pinout! The current pinout was chosen to eliminate the chance of damage if a user accidentally plugs an SRXL2 device into a Spektrum X-Bus connector, which also uses a ZH4. The deprecated pinout may be found in Section 8.3 of this document.

Standard 3-pin Servo Connector:

Pin 1 – SRXL2 Tx/Rx Signal (+3.3V/GND)

Pin 2 – Servo Bus voltage (3.5V – 8.4V)

Pin 3 – Ground / Return



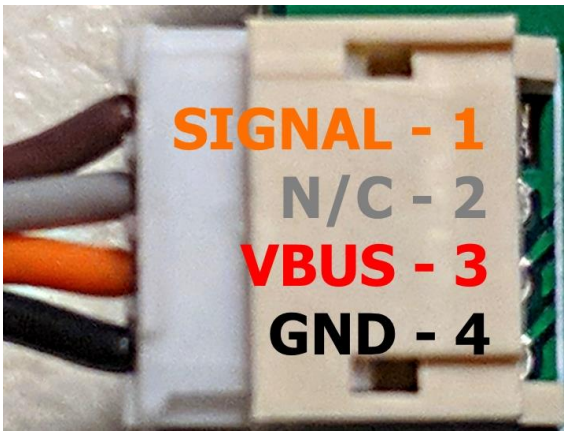
JST ZH4 Connector:

Pin 1 – SRXL2 Tx/Rx Signal (+3.3V/GND)

Pin 2 – Not Connected

Pin 3 – Servo Bus voltage (3.5V – 8.4V)

Pin 4 – Ground / Return



6 HARDWARE-LEVEL PROTOCOL

6.1 UART Configuration

- Baud options configurable on device
  - 115,200bps (default)
  - 400,000bps
- 8 Bits
- No Parity
- 1 Stop (8N1)

6.2 Communication Rules

Communications on the SRXL2 bus generally follow the MPX specifications. Changes are:

a) The bus is not driven while idle.
b) The bus master shall be the RF telemetry receiver or remote receiver with the lowest device ID on the bus (see section 7 for details). Once communications are established, it shall speak a single Control Data packet onto the bus at the same rate as RF packets are received (typically every 5.5ms or 11ms).
c) Only one bus master is allowed on the bus.
d) Secondary remotes may be allowed, but shall follow the rules of other non-master devices. These secondary remotes shall either be a telemetry remote operating in a secondary mode, or a device to interface a standard DSMX remote to the bi-directional bus.
e) All devices shall listen for communications from the bus master for synchronization.
f) Any non-master device can only talk on the bus after an idle time of 2 characters has elapsed, and only if the bus master did not request a reply from anyone via Control Data packet.
g) Devices which transmitted during the previous window should remain silent unless explicitly requested to respond by the bus master to prevent monopolizing the bus.
h) All devices shall remain silent rather than write a message which would conflict with an incoming Control Data packet. The device must look ahead to determine if there is time to send a packet (including the 2-character idle time required between packets).
i) In the case of a missing packet from the bus master, the bus shall remain idle for 50ms in order to allow re-sync or new master selection.
j) New master selection is beyond the scope of this document at this time, but it shall be limited to other receivers or remotes.
k) Spektrum SRXL2 messages shall not exceed 80 bytes total, including 3-byte header and 2-byte CRC. This is to allow use of limited-size DMA buffer transfers.
l) If a message on the bus is irrelevant to the device receiving it, the device should not attempt to reply in order to allow another relevant device the “bus space” to reply.

7 SOFTWARE PROTOCOL

7.1 Overview

The latest Spektrum SRXL2 ID is 0xA6. Packets have varying lengths, with a minimum size of 5 bytes (3 header bytes, 0 data bytes and 2 CRC bytes) and a maximum size of 80 bytes. This protocol is only for products developed after June 2018. For older Spektrum SRXL implementation notes, please refer to Appendix 8.2 on Deprecated Protocols.

All Spektrum SRXL packets begin with a 3-byte header that contains the SRXL Manufacturer ID (0xA6), a Packet Type, and a packet Length in bytes. The payload data (0 to 75 bytes) for that packet type follows that header, and the packet ends with a 16-bit XMODEM CRC in big endian byte order.

<0xA6><Packet_Type><Length><Data_bytes><CRC_Hi><CRC_Lo>

The CRC is computed over all preceding packet data. See Appendix 8.1 for CRC Computation details.

7.1.1 Device IDs

Every device on the SRXL2 bus has a unique Device ID assigned to it. This Device ID is a single byte in which the 4 most significant bits contain a Device Type (see Table 2 below), and the lower 4 bits contain a Unit ID that is used to distinguish multiple devices of the same Device Type:

Device Type	Device ID range	Description	Default ID
0	0x00	No device specified	--
1	0x10 – 0x1F	Remote receiver	0x10
2	0x21 – 0x2F	Receiver	0x21
3	0x30 – 0x3F	Flight Controller (FC)	0x30
4	0x40 – 0x4F	ESC	0x40
5	0x50 – 0x5F	reserved	--
6 – 7	0x60 – 0x7F	SRXL Servo	0x60
8	0x81 – 0x8F	VTX	0x81
9 – 14	0x90 – 0xEF	reserved	--
15	0xFF	All devices (Broadcast)	--

Table 1 - Device Types

The Device ID uniquely identifies a device on a given SRXL2 bus, allowing the bus master to request a response from a specific device on the bus. This is used to prevent bus collisions when multiple slave devices are present. Device ID 0xFF is used as a broadcast request for all devices to receive, and 0x00 is used to indicate no devices (i.e. if a reply packet is not desired).

When the **Unit ID** portion (the lower nibble) of the Device ID is 0, this tells the device to send a Handshake packet during startup without being polled by a bus master. The Default IDs given above are chosen because typically Receivers and VTXs should NOT talk on the bus until requested.

NOTE: Only one device on the bus should have the Unit ID set to 0 to avoid handshake collisions.

In most current applications, an SRXL2 bus will not have more than one device of each device type on the bus, so the Default ID listed in Table 1 for a given Device Type can be used as the Device ID. In situations where multiple devices of the same type share the bus, the Unit IDs should be sequentially numbered, generally starting from the Default ID value. For example, if there is one Receiver (acting as the bus master) and four Smart ESC devices on the bus, the Receiver will be assigned Device ID 0x21, and the ESCs should be assigned Device IDs 0x40 thru 0x43. The different Unit IDs for the ESCs could be set via physical switches or jumpers, or through a configuration value stored in non-volatile memory on the device.

Note: If there is enough demand for it, we can add support for a scheme to set the Unit ID of a device via SRXL2 that would require attaching the devices to the bus one at a time during a special programming mode. For now, this is not implemented, so physical switches or a stored configuration value are preferred.

7.1.2 Packet Types

The following table contains a list of Packet types:

Packet Description	Packet Type	Total Byte Length
Handshake	0x21	14
Bind Info	0x41	21
Parameter Configuration	0x50	14
Signal Quality	0x55	10
Telemetry Sensor Data	0x80	22
Control Data	0xCD	5 to 80

Table 2 - Packet Types

Details of each of these packets are described in the following subsections.

NOTE: In the following packet descriptions, all payload data elements larger than a single byte use **little endian** byte order unless explicitly stated otherwise (as with the big endian CRC, for example).

7.2 Handshake Packet

The Handshake packet is sent during startup by each device on the bus, and must **always** be sent at the default baud rate of 115200. By following the protocol described in section 7.2.1 below, this will allow the bus master to poll each device on the bus to determine their requested telemetry priority and to set the fastest baud rate allowed by all devices.

The packet format is as follows:

<0xA6><0x21><Length><SrcID><DestID><Priority><BaudRate><Info><UID><CRC>

Length: 14 (3 byte header, 9 bytes of data, and 2 CRC bytes).

SrcID: A single byte containing the Device ID of the device sending the Handshake packet. See Section 7.1.1 for details on Device IDs.

DestID: A byte containing the Device ID of the destination device that should respond to this Handshake packet. When a slave device is announcing their presence to the bus master during startup (which happens if the lower nibble of the SrcID is 0), this DestID should be set to 0. When the handshake is sent by the bus master to command everyone to set their baud rate, this is set to 0xFF.

Priority: A byte value from 1 to 100 sent by a slave device to the bus master to indicate how often it would like the bus master to request its Telemetry data, with higher values indicating higher priority. The bus master is free to use this information in whatever manner it deems appropriate, and may adjust as needed. A typical value for one telemetry packet is 10.

BaudRate: A byte containing the supported SRXL2 baud rates of the device, selected from the following two values: 0 = 115200 baud only, 1 = 400000 baud supported.

Info: A byte used as a bitfield to report additional device options that could be useful for an FC or Smart ESC. Currently, bit 0 = 1 if the device can send long-range telemetry, or 0 if short-range only.

UID: A 32-bit unique identifier to allow detection of multiple devices on the bus erroneously set to the same Device ID. If more than one device responds simultaneously with a handshake, this unique number will ensure that the CRC will be invalid. This value can be randomly generated for each device or can be a hash based on a serial number – the only requirement is that it be statistically unlikely for two devices to be assigned the same value.

7.2.1 Handshake Protocol

The Spektrum SRXL2 Handshake process is intended to take place upon device reset to allow a bus master to identify all other devices on the bus and negotiate a baud rate with them. The protocol is intended to handle cases of brown-out or momentary power loss on any of the individual devices including the bus master.

If the bus master is a Receiver that shares the SRXL2 UART line with a throttle PWM line, it will refrain from sending a handshake until it has received one from a slave device or until 200ms has elapsed. This is intended to prevent a Receiver from sending an SRXL2 handshake packet to a non-SRXL ESC or FC, possibly resulting in unintended movement or erratic behavior. If no handshake is received after 200ms, the Receiver assumes that no SRXL2 devices are present and defaults to PWM

mode. If the SRXL2 communications are on a dedicated UART pin then the bus master can be configured to wait indefinitely for a device to initiate communications or can initiate the handshake itself by polling specific devices.

The following basic steps describe the handshake process all non-master devices should follow after a power-on or brown-out reset:

1. Configure the UART to 115200 baud.
2. Listen for 50ms after reset for serial activity. If non-handshake data is received, skip the handshake process, determine the current baud rate, and resume communications.
   (This allows a slave device to resume communications if a brown-out reset occurred but the bus master did not reset. If UART activity is detected, the slave must use auto-baud calculations or frame error detection to determine the active baud rate before resuming. Note that in order to know which receiver device ID should get telemetry data without handshaking, the destination device ID in the telemetry should be 0xFF – this will request a re-handshake from the master.)
3. If Unit ID is not 0, skip to step 4. Else if UnitID == 0, repeatedly send a Handshake packet with DestID = 0 every 50ms for at least the first 200ms after reset to prompt the bus master to speak. Stop sending and advance to the next step as soon as a Handshake is received.
4. Listen for Handshake packets from the bus master.
5. When a Handshake packet is received where the DestID matches its Device ID, a Handshake packet reply should be sent in response to announce the highest supported baud rate and the requested priority level for the device's telemetry data.
6. When a Handshake packet is received from the bus master with DestID = 0xFF, all devices must configure their UART baud rate to the rate contained in that Handshake packet so that normal SRXL2 communications can begin.

If the bus master ever stops sending during normal operation and 50ms elapse with no packets, all slave devices should return to 115200 baud and restart the above handshaking process since that would indicate a brown-out reset of the bus master.

It is up to the bus master to determine the highest speed supported by all devices before broadcasting a Handshake packet configuring the actual baud rate of the bus. Upon receiving this final broadcast Handshake, all devices should switch their baud rate accordingly.

7.2.2 Handshake example

TODO

7.3 Bind Info Packet

Upon completion of bind the bus master shall issue a Bind Info packet. This packet could be used by other receivers to bind to the designated master without having had to listen to the bind over the air. The Bind Info packet can also be generated by a device on the bus to request that all listeners enter bind mode, or to query the bind status. In these cases, the Bind Info packet is sent instead of Telemetry.

The packet format shall be:

<0xA6><0x41><Length><Request><DeviceID><Type><Options><GUID><UID><CRC>

Length: 21 (length of the payload + overhead).

Request: A single byte from Table 3. Invalid values shall cause no action.

Request type	Value	Details
Enter Bind Mode	0xEB	To Receiver (FC → RX or RX → Remote) to command it to enter normal RF bind mode
Request Bind Status	0xB5	To Receiver to request bind type and GUID
Bound Data Report	0xDB	From Receiver to provide bind type and GUID
Set Bind Info	0x5B	To Receiver(s) to bind to a given type and GUID

Table 3 - Bind Request Options

DeviceID: A byte containing the destination of Bind requests or the source of Bound Data Reports.

Type: A byte containing the bind type according to Table 4:

Bind Type	Value
Not Bound / Exit Bind	0x00
DSM2 1024 22ms	0x01
DSM2 1024 (MC24)	0x02
DSM2 2048 11ms	0x12
DSMX 22ms	0xA2
DSMX 11ms	0xB2
Surface DSM2 16.5ms	0x63
DSMR 11ms / 22ms	0xE2
DSMR 5.5ms	0xE4

Table 4 - Bind Status Types

Options: A byte with 2 RF enable bits: Set bit 0 to allow Telemetry, and set bit 1 to allow Bind reply.

GUID: 64-bit value containing the GUID used by the RF system during bind.

UID: A 32-bit unique identifier from the Receiver. This is different than the GUID used for binding to the transmitter, and could be used to detect a new RF device with the same Device ID.

The Bound Data Report includes the UID of the reporting device. When a Bind Status request is issued to Device ID 0, the bus master should report the bind info with the Bound Data Report packet. The requesting device can then use that information to send a Set Bind Info command to other Receivers or Remote Receivers on the SRXL2 bus to directly bind them to a transmitter without an RF bind packet.

7.4 Parameter Configuration

Devices may allow query and configuration of internal parameters. The parameters and their values are specific to the device and are not enumerated here. This function is intended to allow the configuration of devices by host controllers, such as for a flight controller to specify the units of signal quality, the priority of telemetry messages, etc. It may also be used to configure and query run-time variables within the devices.

The packet format shall be:

<0xA6><0x50><Length><Request>< DestID><ParamID><ParamValue><CRC>

Length: 15 (length of the payload + overhead).

Request: A single byte containing one of the following commands from Table 5:

Request	Value
Query Parameter Value	0x50
Write Parameter Value	0x57

Table 5 – Parameter Configuration Options

DestID: A byte containing the Device ID to which the Parameter Configuration packet is being sent.

ParamID: A 32-bit value which specifies the configuration parameter to be queried or written.

ParamValue: A 32-bit value field containing the parameter value, unused for Query messages. If a ParamID is for an 8-bit or 16-bit value, the full 32-bit field is still used. Signed values shall be sign-extended in the message, and unsigned shall have high bits of 0.

In all messages, data is transferred as 32-bit values. The host and device may thus have independent word sizes with no specific knowledge of how the data is stored on the other side.

The definition of ParamIDs is outside the scope of this document, and is device-specific.

7.5 Signal Quality Packet

Devices with RF receivers in them may support signal strength data upon request from the host. Note that this is intended for a flight controller to request this in lieu of sending a telemetry packet, or for a bus master to be configured to send quality status reports regularly in addition to its control data.

The packet format shall be:

<0xA6><0x55><Length><Request><AntennaA><AntennaB><AntennaC><AntennaD><CRC>

Length: 10 (length of the payload + overhead).

Request: A single byte according to Table 6. Invalid values shall cause no action.

Request	Value
Request Quality Status	0x52
Quality Status Report	0x53

Table 6 – Signal Quality Options

The four Antenna fields shall be signed 8-bit integers. A value of 0 shall indicate that there is no data available for the given antenna. A positive value shall indicate that the units are RSSI percent. A negative value shall indicate the actual value in dBm. Note that the units preferred by the host may be configured in the RF receiver device using the parameter control messages if supported by the RF receiver.

7.6 Telemetry Sensor Data Packet

The Telemetry Sensor Data packet includes a complete 16-byte telemetry packet as it would be delivered over the RF as if it had come from an I^2C device on the X-Bus. This will enable use of existing telemetry devices by plugging them into an adapter interface (which would handle enumeration and SRXL2 protocol conversion), or by new sensors which utilize the SRXL2 protocol natively.

The packet format shall be:

<0xA6><0x80><Length><DestID><16-byte telemetry packet><CRC>

Length: 22 (length of the 16-byte telemetry packet + overhead).

DestID: A byte containing the Device ID of the receiver that should transmit telemetry over RF. When set to 0, this disables ALL telemetry transmits from receivers on the bus. The intention is to allow flight controllers or base receivers to use this to select the best receiver to send telemetry data at any given time. If the telemetry device did not receive the Handshake and does not know the Device ID of the bus master who requested telemetry, this should be set to 0xFF to request a Handshake packet.

Telemetry packet: Refer to the Telemetry Sensor Struct .h file mentioned in Section 4 for the various types of telemetry data.

NOTE: Telemetry packets are sent by the device matching the ReplyID sent in a Control Data command. If the device supports multiple telemetry packets, it is up to the device to prioritize and decide which single telemetry packet to send each time a reply is requested.

7.7 Control Data Packet

The Control Data Packet type can be used to provide various forms of control data to slave devices. The packet format shall be:

<0xA6><0xCD><Length><Command><ReplyID><Payload><CRC>

Length: Varies depending on the Payload data being sent – see details below.

Command: A byte consisting of value from the following table that describes the type of data included in the payload:

Command	Value
Channel Data	0x00
Failsafe Channel Data	0x01
VTX Data	0x02
reserved	0x03

Table 7 – Control Data Commands

ReplyID: A byte containing the Device ID of the device from which the bus master wants to receive a reply. If no reply is desired, set this to 0x00 (no device). Setting this to 0xFF (all devices) should be avoided since bus collisions would occur when more than one device tries to reply at once.

It is important to remember during these messages that the slave device should only reply if the ReplyID matches and the message is relevant to it (see Section 6.l).

For example, if the receiver bus is connected to both a flight controller and VTX, the flight controller should not reply to a VTX Data command (even if the ReplyID matches) to allow the VTX the opportunity to reply instead with its current parameter status as a VTX telemetry message. If the bus is only connected to a flight controller which is configured to operate a VTX connected to some other pins, then the flight controller would be allowed to reply in this scenario since it is a middle man to the receiver and VTX, and therefore can take the role of the VTX on the bus.

Payload: The number of payload bytes are described in the reminder of this section. Note that Channel Data and Failsafe Channel Data both use the same basic payload data format.

7.7.1 Channel Data Payload

The channel data payload consists mainly of updated servo values sent by the bus master to all devices on the bus during normal operation. Channels not sent in a packet should hold their last value.

```
typedef struct
{
      int8_t       rssi;
      uint16_t     frameLosses;
      uint32_t     channelMask;
      uint16_t     channelData[# channels];
} CH_DATA_PAYLOAD;
```

rssi: A signed byte containing the signal quality, where a positive value indicates that the units are RSSI percent, and a negative value indicates the actual value in dBm. Note that the units preferred by the host may be configurable using the parameter control messages if supported by the RF receiver.

frameLosses: A 16-bit unsigned value containing the total frame loss count for this Receiver.

channelMask: A 32-bit mask to indicate which channels are contained in the channelData that follows, where bit 0 (lsb) corresponds to Channel 1 and bit 31 (msb) corresponds to Channel 32. During normal operation, any channels not included in the Channel Data packet should hold their last value.

channelData: An array of *N* 16-bit channel data values, where *N* is the number of set channelMask bits, ranging from 0-32. Each channel data value is sent as an unsigned 16-bit value from 0 to 65532 (0xFFFC), with 32768 (0x8000) representing "Servo Center". The channel value must be bit-shifted to the right to match the application's accepted resolution. For example, to match DSMX 2048 data (11-bit), shift each value 5 bits to the right. The lower 2 bits are reserved for future use and must be 0.

Packet Example:

<0xA6><0xCD><0x1C><0x00><0x30><0x58, 0x0B, 0x00, 0x37, 0x06, 0x00, 0x00, 0xA0, 0x2A, 0x00, 0x80, 0x04, 0x80, 0xFC, 0x7F, 0x54, 0xD5, 0xA0, 0x2A, 0xA0, 0x2A><CRC>

This sample packet indicates the following:

Channel Data with a request that the Flight Controller (device ID 0x30) send a telemetry reply.

Current RSSI value of 88%, with a total of 11 lost frames.

The payload includes the following channels:

0x00000637 = 0b00000000000000000000011000110111 = CH 1, 2, 3, 5, 6, 10, 11

The data for the channels is as follows:

CH 1 = 0x2AA0 = 10912 (approx -100% on Spektrum transmitter)
CH 2 = 0x8000 = 32768 (center position)
CH 3 = 0x8004 = 32772 (1 tick above center at 14-bit resolution)
CH 5 = 0x7FFC = 32764 (1 tick below center at 14-bit resolution)
CH 6 = 0xD554 = 54612 (approx. 100% on Spektrum transmitter)
CH 10 = 0x2AA0 = 10912
CH 11 = 0x2AA0 = 10912

7.7.2 Failsafe Channel Data Payload

The failsafe channel data payload consists mainly of servo values sent by the bus master to all devices on the bus when RF communications have been lost and failsafe values must be sent. These failsafe values are usually captured during bind and are stored in non-volatile memory on the Receiver. Unlike the normal Channel Data payload, channel values not included in the Failsafe Channel Data payload should be disabled rather than holding their last value.

```
typedef struct
{
      int8_t       rssiMin;
      uint16_t     holds;
      uint32_t     channelMask;
      uint16_t     channelData[# channels];
} FS_CH_DATA_PAYLOAD;
```

rssiMin: A signed byte containing the lowest possible signal quality for the Receiver, where a positive value indicates that the units are RSSI percent, and a negative value indicates the actual value in dBm. Note that the units preferred by the host may be configurable using the parameter control messages if supported by the RF receiver.

holds: A 16-bit unsigned value containing the total number of holds for this Receiver since power-up.

channelMask: A 32-bit mask to indicate which channels are contained in the channelData that follows, where bit 0 (lsb) corresponds to Channel 1 and bit 31 (msb) corresponds to Channel 32.
NOTE: In Failsafe Data, channels that are not included in this packet should be disabled instead of holding their last value.

channelData: An array of *N* 16-bit channel data values, where *N* is the number of set channelMask bits, ranging from 0-32. Each failsafe channel data value is sent as an unsigned 16-bit value from 0 to 65532 (0xFFFC), with 32768 (0x8000) representing "Servo Center". The lower 2 bits are reserved for future use, and will be 0 for now.

Details

When a receiver consecutively misses too many frames of RF data, it enters failsafe mode. Instead of sending the latest channel data that it received from the transmitter, it will send failsafe values (captured during bind) for all channels that should still be driven to a given value. Any channel that is not sent in the failsafe packet should be disabled rather than driven to a given position or throttle value.

It is ultimately up to the ESC or Flight Controller to decide what to do with these failsafe values, but it is strongly recommended that they be used in this manner to provide a consistent experience during RF signal loss.

7.7.3 VTX Data Payload

The VTX data payload command will be similar to that used for telemetry:

```
typedef struct
{
      uint8_t      band;
      uint8_t      channel;
      uint8_t      pit;
      uint8_t      power;
      uint16_t     powerDec;
      uint8_t      region;
} STRU_TELE_VTX;
```

band: VTX Band (0 = Fatshark, 1 = Raceband, 2 = E, 3 = B, 4 = A, 5-7 = Reserved)

channel: VTX Channel (0-7)

pit: Pit/Race mode (0 = Race, 1 = Pit). Race = (normal operating) mode. Pit = (reduced power) mode. When PIT is set, it overrides all other power settings.

power: VTX Power (0 = Off, 1 = 1mw to 14mW, 2 = 15mW to 25mW, 3 = 26mW to 99mW, 4 = 100mW to 299mW, 5 = 300mW to 600mW, 6 = 601mW+, 7 = manual control)

Note: This **power** value has priority over **powerDec** if both are set. **power** must be 0xFF for **powerDec** to be used. If both **power** and **powerDec** are 0xFF/0xFFFF, then no power change should occur

powerDec: VTX Power as a decimal 1mw/unit. If set to 0xFFFF, it should be ignored and **power** should be used instead. If **powerDec** is used to set a power level not supported by the VTX, the VTX should set its power to the next power level below that specified **powerDec**.

region: Region of operation (0 = USA, 1 = EU)

When given invalid data for the fields described, the behavior of the VTX system is determined by the Flight Controller or VTX.

Packet Example:

<0xA6><0xCD><0x0D><0x81>< 0x01, 0x03, 0x00, 0x02, 0xFFFF, 0x00 ><CRC>

This packet indicates the following:

VTX Data with a telemetry reply requested.
Band = 0x01 = Raceband
Channel = 0x03 = Channel 4 (Assuming user channel range = 1 to 8)
Pit = 0x00 = Race/Normal operation
Power = 0x02 = 15mW to 25mW
PowerDec = 0xFFFF = use value indicated by Power
Region = 0x00 = USA

Assume this packet type has a variable length as new parameters may be added in the future.

8 APPENDICES

8.1 CRC Computation

The SRXL protocol uses a standard 16-bit CRC commonly referred to as XMODEM or ZMODEM CRC. The CRC uses the polynomial 0x1021 (which corresponds to $x^{16} + x^{12} + x^5 + 1$), with an initial value of 0, and operates with no bit reflection on the input or output (i.e. computations are performed on data assuming the most significant bit is first). Note that some hardware CRC engines might need to be configured to reflect the bits if they operate on the data as it comes in on the wire since UARTs transmit the least significant bit first.

The following code samples are included (with slight modifications) from the external Multiplex document "SRXL Specification Version 2.9", page 2.

8.1.1 CRC Algorithm (in C)

```
uint16_t Crc16(uint16_t crc, uint8_t data)
{
    crc = crc ^ ((uint16_t)data << 8);

    for(int i = 0; i < 8; ++i)
    {
        if(crc & 0x8000)
            crc = (crc << 1) ^ 0x1021;
        else
            crc = crc << 1;
    }

    return crc;
}
```

8.1.2 Sample CRC Usage (in C)

```
uint8_t rxBuffer[80];  // Assume this contains received SRXL packet
uint8_t rxLength;      // Assume this contains total length in bytes of received packet
uint16_t computedCRC = 0;

for(uint8_t i = 0; i < rxLength - 2; ++i)
{
    computedCRC = Crc16(computedCRC, rxBuffer[i]);
}

uint16_t rxCRC = ((uint16_t)rxBuffer[rxLength – 2] << 8) | (rxBuffer[rxLength – 1]);

if(computedCRC == rxCRC)
    // Success!
else
    // Failure!
```

8.2 Deprecated Protocols

8.2.1 Unidirectional SRXL (ID 0xA2)

Obsolete since May 2016.

This SRXL packet format was used by AR9020 (and similar vintage remote receivers), and used a fixed size packet to transfer incoming RF data. Telemetry data was not supported. The RF data format was the 16-byte data received over the air with a 2-byte CRC that was incorrectly computed (TODO: add details after researching).

8.2.2 1st Gen Bi-directional SRXL (ID 0xA5)

Not for use in new designs as of July 2018.

This ID was first used by the AR7700, using the exact same fixed 18-byte format as the Unidirectional protocol in section 8.2.1 above, but with corrected CRC computation.

Bi-directional communication was added for the SPM4649T telemetry receiver. In order to differentiate it from the AR7700, the two CRC bytes are swapped (i.e. sent in little endian order). This allowed the connected SRXL slave device to determine whether bi-directional SRXL was supported.

In the SPM4649T, SRXL was used as a hybrid system, with RF data still being transferred to the slave device using the remote receiver protocol, and only Telemetry Sensor and Enter Bind Mode packets being sent from the slave to the telemetry receiver over SRXL. The packets were sent based on the phase bit of the RF data since the Channel Data packet that would normally request this was not being sent. Unfortunately, the CRC computation for these two commands incorrectly excluded the first 4 bytes of packet data and the final packet data byte from the CRC computation. This error prompted the change to a new SRXL ID.

See **Rev F** of this document for more details.

8.3 Deprecated Connector Pinout

NOTE: This pinout is ONLY used on the SPM4649T – DO NOT USE ON OTHER SRXL DEVICES!

Connection to the SPM4649T uses a JST ZH4.

Pin 1 = Vbus (3.5-8.4V)

Pin 2 = Common/Ground/Return

Pin 3 = signal (3.3V logic, high impedance)

Pin 4 = N/C

REVISION HISTORY

Rev	Date	Author	Description
P0	2016-05-25	AK/TK/ MFA	For initial review.
P1	2016-07-20	MFA	Changed bind message length to 0x13. Added pinout labels for ZH4 connector
P2	2016-10-28	MFA/AK	Limit implementation status to *Enter Bind Mode* command.
A	2016-11-10	MFA	Typos, release.
B	2016-12-15	MFA	Added legal info
C	2016-12-20	MFA	Fixed bind packet length
D	2017-08-31	MFA	Fixed recommended bind request type and revision in footer
E	2018-01-09	AK	Add Signal Quality support, Parameter Configuration
F	2018-03-30	MFA	Added Control Data packet. Changed length to 80 bytes. Added new baud rate option.
G	2018-09-18	MWO	Change SRXL id to 0xA6 to correct CRC issues. Add section on Device IDs and switch Control Data to use it. Add Handshake packet. Change bind type definitions to add Surface bit. Move old protocol descriptions to Appendices. Change Channel Data to reserve least significant bits and use full 16-bit values to ease 3rd party use. Change pinout of ZH4 connector and add servo connector.
H	2018-10-26	MWO	Add “info” byte to Handshake packet to report device options. Add “options” byte to Bind packet containing enable bits to control RF transmission of telemetry and bind replies. Add destination DeviceID to Telemetry packet so that only the selected device is allowed to send telemetry over RF.
I	2019-02-27	MWO	Add page for Failsafe Channel Data. Move Telemetry to its own page for visibility.
J	2019-06-24	MWO	Rename to SRXL2. Change NOT_BOUND value to 0x00. Add telemetry destDevID = 0xFF to request handshake.