

Description of the communication protocol for the Positive Grid Spark amplifiers

By Paul Hamshire with a great acknowledgement to Yury Tsybizov (ytsibizov) and Justin Nelson (jrnelson). Thanks to Ian McKellar for pointing out the MIDI SysEx similarity and that the packed data format is msgpack (www.msgpack.org)

Overview of communication

The Spark amplifiers communicate with the Spark app over Bluetooth. It uses 'serial bluetooth' or 'classic bluetooth' for the Android app and 'BLE' for iOS.

The app sends messages to change preset, change an effect, change the parameter for an effect (eg gain). It can also request the details of each hardware preset, the name of the amp and the serial number.

In return, the amp will send messages when one of the presets is changed or when a knob is moved. This allows the app to mimic the settings on the amp at all points.

When the app starts, it asks the Spark for its name, serial number and all four hardware presets.

Then communication is event driven - either from the app or the amp.

Overview of message format

The bluetooth messages are exchanged in a specific data format. The terminology below is one I created to help understand the underlying structure.

Messages are exchanged in blocks. Each block contains one or more chunks. Each chunk contains data - which is all, or part of, the message.

Blocks and chunks appear to have size limits which means: messages span chunks, and chunks span blocks.

The simple messages are from the app to the amp, and are usually just one block, one chunk and the data.

Sending a preset, or receiving a preset, is more complex and involves multiple blocks and chunks.

Figure 1 shows the relationship between the blocks, chunks and data that make up the message.

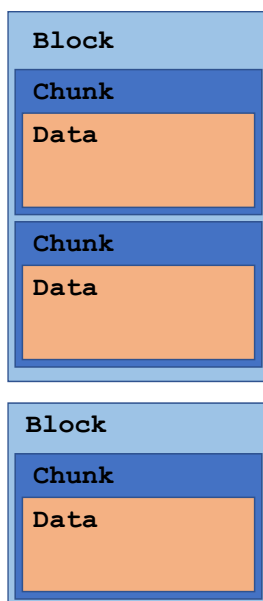


Figure 1

When the app sends a message then the Spark (usually) responds with an acknowledgement message.

Blocks sent to the amp seem to have a maximum size of 0xad.

Blocks sent from the amp seem to have a maximum size of 0x6a.

Block format

Each block has a header and then contains the chunk / data.

| Offset | Length | Description |
|--------|--------|---|
| 0 | 4 | 0x01fe0000 |
| 4 | 2 | Direction of the message: 0x41ff - from Spark 0x53fe - to Spark |
| 6 | 1 | Size of this block (including this header) |
| 7 | 9 | Zeros |
| 10 | | The chunk / data |

Figure 2 shows an example of a block header.

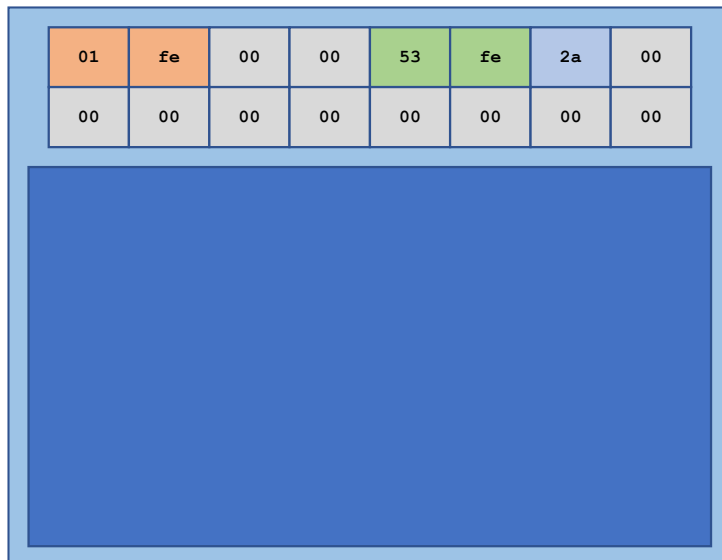


Figure 2

Chunk format

| Offset (in block) | Length | Description |
|-------------------|--------|----------------------|
| 10 | 2 | 0xf001 |
| 12 | 1 | Sequence number |
| 13 | 1 | Checksum (8 bit Xor) |
| 14 | 1 | Command |
| 15 | 1 | Sub-command |
| 16 | | Data |
| | 1 | 0xf7 |

The chunk starts with fixed bytes of 0xf001 and ends with the byte 0xf7. This is very like the MIDI SysEx wrapper of 0xf0 and 0xf7.

The header includes a sequence number which increments with each message (so it remains consistent across chunks and blocks for the same message). When the amp acknowledges a message it contains the sequence number in the acknowledgement message.

The checksum is an 8-bit xor checksum of the data part - it excludes the chunk header and the f7 trailer.

The command and sub-command describe what the change is to the amp or from the amp (eg change gain on the amp model).

Figure 3 shows a block header, chunk header and trailer.

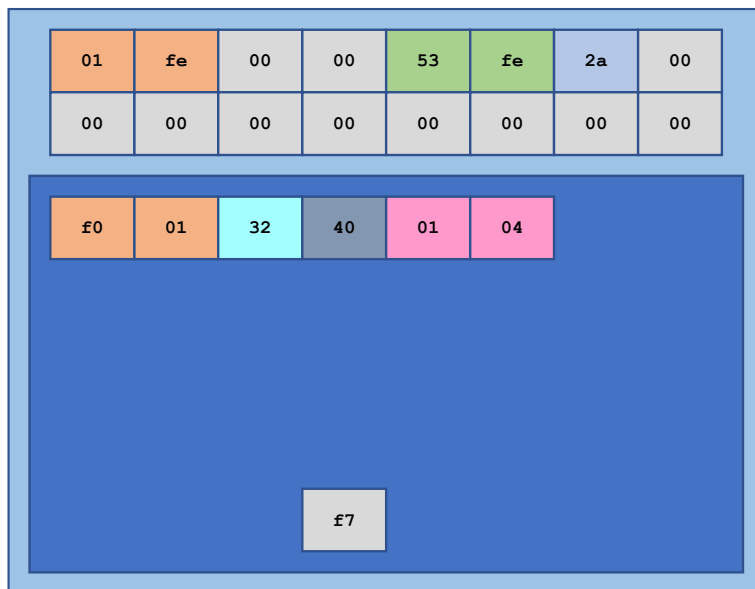


Figure 3

Data format

The message is a sequence of variables. Each variable has a distinct pattern which identifies it.

The variables are stored in the data section in sequences of 8 bytes. In the data section bytes have the top bit set to zero, so only carry 7 bits of data. The remaining 8th bit is packed into another byte which only contains the 8th bit of each of the bytes in the sequence.

So the format is the special byte containing the 8th bits, followed by seven data bytes.

Figure 4 shows the structure of the sequence - the '8th bits' byte followed by up to 7 data bytes.

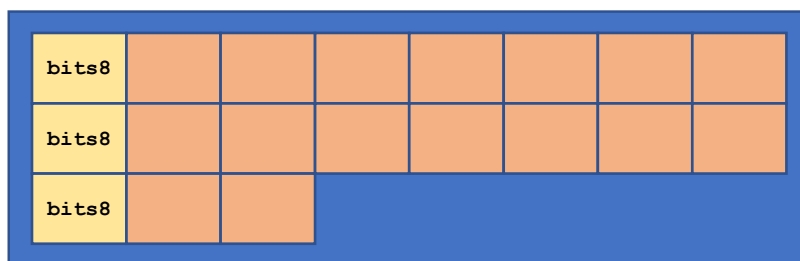


Figure 4

Figure 3 shows an example of the mapping for the missing 8th bit. In this example, the data in the fourth data byte in the first sequence should have its 8th bit added back (represented by bit 3 being set in the '8th bits' byte. And the same for the third and sixth bytes in the second sequence (bits 2 and 5 set in the '8th bits' byte).

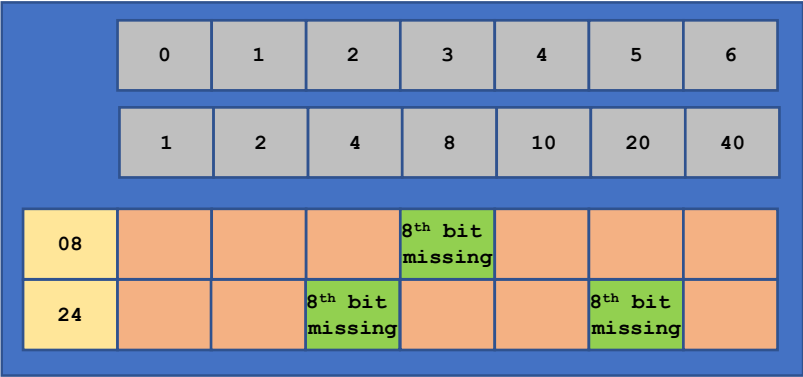


Figure 5

To interpret the data it is therefore essential to add back these bits.

Overall structure

Figure 6 shows a representation of the overall structure, including headers, trailers and format bytes. Figure 7 shows an example of the headers and footers. These both show a single block / single chunk message and summarise the description so far.

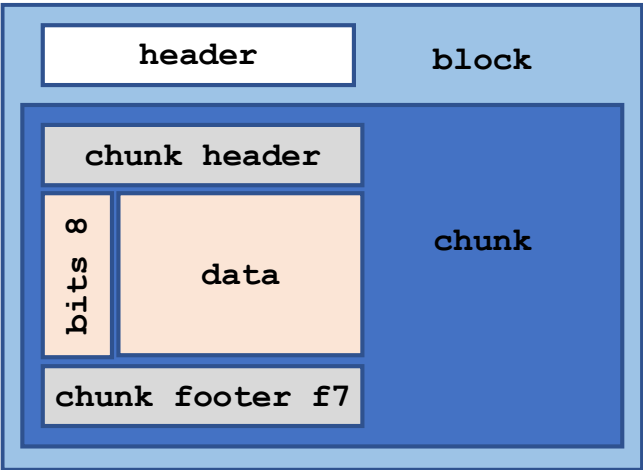


Figure 6

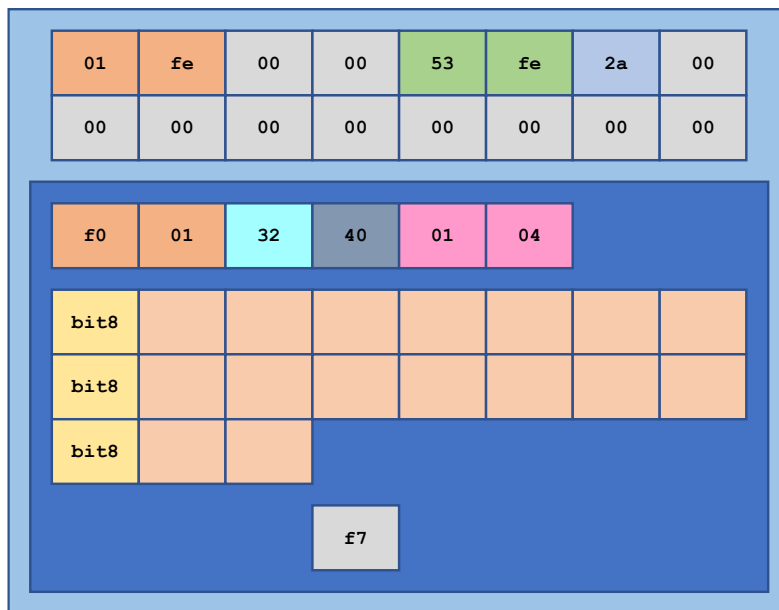


Figure 7

Variable types

The data format is based on msgpack (www.msgpack.org).

The data in the message is a set of variables - integers, strings, Booleans and floating-point values.

| Type | Length | Description | First byte range |
|--|--------|--|------------------|
| Short Integer | 1 | Data value from 0x00 to 0x07f. How is this distinguished from 0x00 as the start of an integer. How is this distinguished from the first byte of the alternative short string? | 0xcc 0x80 - 0xff |
| Integer | 2 | Data value from 0x80 to 0xff. Prefixed by 0xcc. | 0xcc 0x80 - 0xff |
| Fixed array size | n+1 | Data value from 0x00 to 0x0f, stored as data value + 0x90. | 0x90 - 0x9f |
| Short string (1-31 characters) | n+1 | First byte is the length + 0xa0, then the bytes of the string in ASCII encoding | 0xa0 - 0xbf |
| Alternative short string (1-31 characters) | n+2 | First byte is the length, next byte is the length + 0xa0, then the bytes of the string in ASCII encoding (Unsure if this is limited to 15 characters but it would be logical given the apparent use of the first byte to describe the data type.) | 0x01 - 0x1f |
| Long string | n+2 | First byte is 0xd9, then the length, then the bytes of the string | 0xd9 |
| Boolean Off | 1 | A single byte representing effect Off | 0xc2 |
| Boolean On | 1 | A single byte representing effect On | 0xc3 |
| Float | 5 | A float value - 4 bytes big endian with a preceding byte of 0xca | 0xca |

Figure 8 shows these data types in a visual format

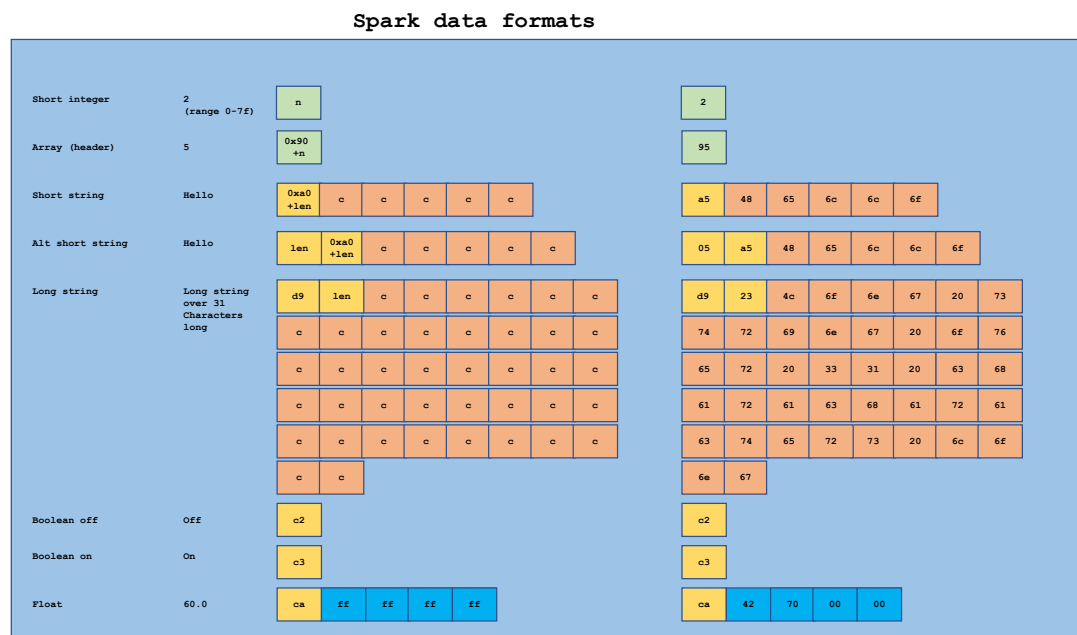


Figure 8

Figure 9 shows a completed message, with all headers, footers, data and format bytes.

This is data with a string "LA2AComp", a short integer of 1 and a float represented by 0x3f4d42c4 (with the 8th bit added back to the final 0x44)

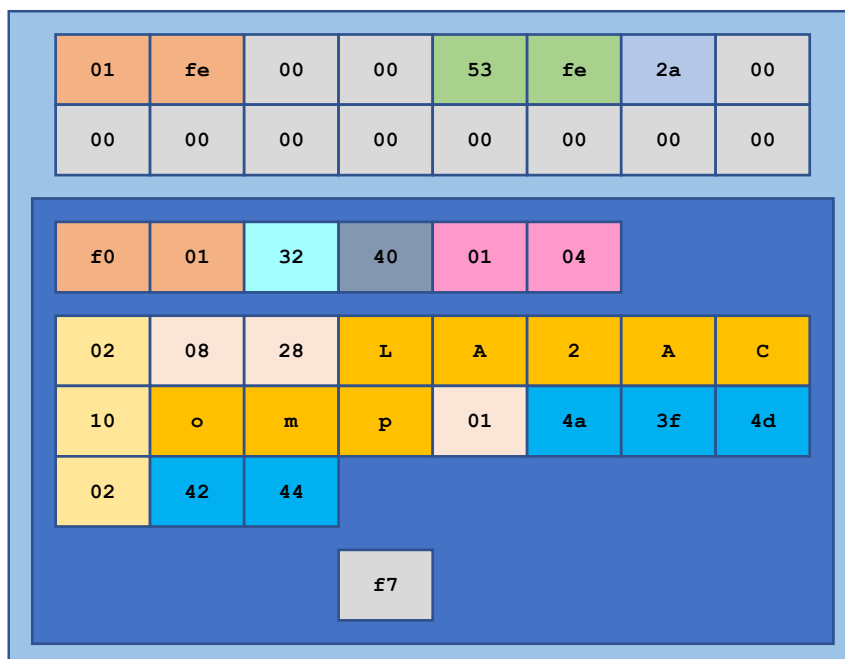


Figure 9

Figure 10 shows this with explanatory labelling.



Figure 10

Float representation

Floats are based on the 4-byte IEEE-754 encoding. (As with all the other data section formats, the bytes are 7-bit only and the missing 8th bits are in the first byte of any 8 byte sequence.)

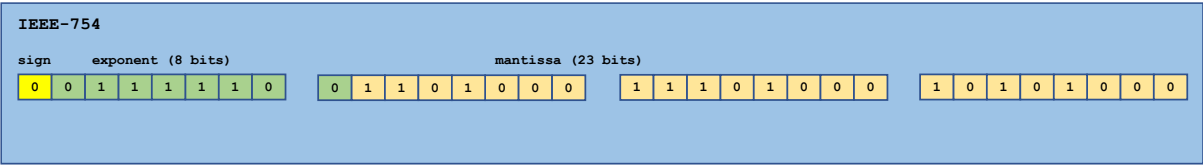


Figure 11

Special values are shown below

Effects with toggle switches (such as Multi Head delay)

| Hex | Float | Special meaning |
|-------------|-------|-----------------------------|
| 00 00 00 00 | 0.0 | False (for a toggle switch) |
| 3f 80 00 00 | 1.0 | True (for a toggle switch) |

Digital delay

| Hex | Float | Special meaning |
|-------------|-------|-----------------|
| 3f 38 51 ec | 0.72 | 1s |
| 3f 19 99 9a | 0.60 | 500ms |
| 3e cc cc cd | 0.40 | 200ms |
| 3e 99 99 9a | 0.30 | 50ms |

Multi Head delay

| Hex | Float | Special meaning |
|-------------|-------|-----------------|
| 00 00 00 00 | 0.00 | Head 0 + 2 |
| 3e b3 33 33 | 0.35 | Head 0 + 1 |
| 3f 26 66 66 | 0.65 | Head 1 + 2 |
| 3f 73 33 33 | 0.95 | Head 0 + 1 + 2 |

Messages that span chunks and blocks

The only messages large enough to span multiple chunks and blocks are those sending a complete preset, either to or from the amp. They can be identified by the command and sub-command (see later).

In these cases, the first three bytes of the data (after the format byte) represent which sub-chunk this is.

The size of the chunks and the data in these bytes depends on the direction of the message.

Multi-chunk messages sent to the amp

In this case, whilst the message spans multiple chunks, each chunk fills a block. The maximum sending block size is 0xad bytes, so the size of the chunk is 0x9b.

This is calculated as block size - block header - chunk header - chunk trailer (0xad - 0x10 - 0x06 - 0x01 = 0x9b)

The first four bytes of the chunk data are as in the table below - representing the format byte and the multi-chunk sub-header.

| Offset (in chunk) | Length | Description |
|-------------------|--------|--|
| 6 | 1 | First '8 th bits' byte |
| 7 | 1 | Total number of chunks |
| 8 | 1 | Reference number of this chunk (0 to total number of chunks - 1) |
| 9 | 1 | Size of this chunk (in data bytes which therefore excludes counting the '8 th bit' bytes, max 0x80) |

The number of data bytes remaining is a count of useful data bytes - total bytes less the '8th bit' bytes.

Figure 12 shows the overall structure of a multi-chunk message sent to the amp.

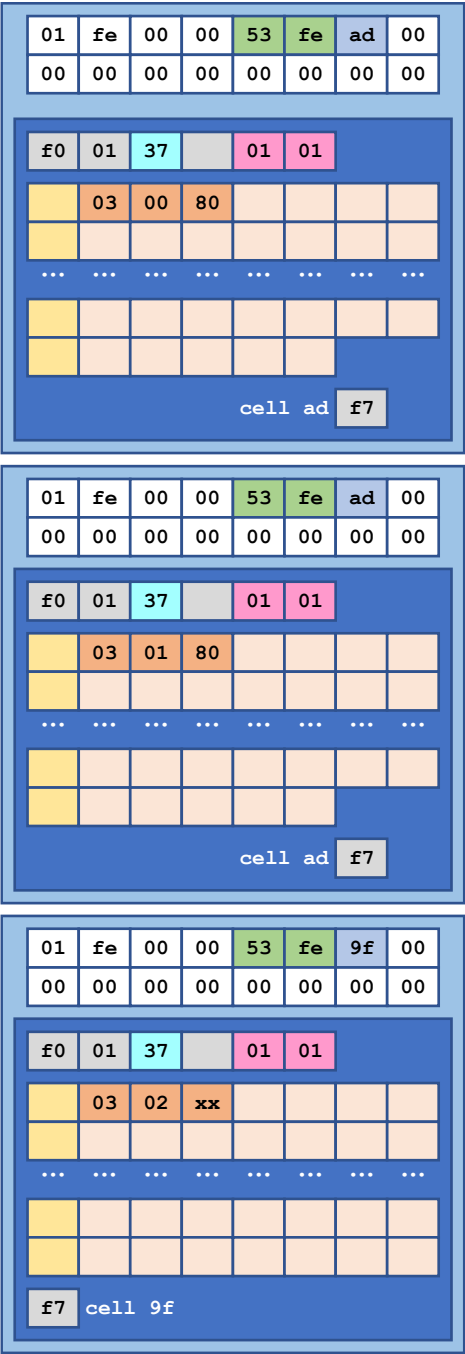


Figure 12

Multi-chunk messages received from the amp

In this case, whilst the message spans multiple chunks, there are multiple chunks in each block. Each chunk has a maximum size of 0x27 and the block has a maximum size of 0x6a.

The first four bytes of the chunk data are as in the table below - representing the format byte and the multi-chunk sub-header.

| Offset (in chunk) | Length | Description |
|-------------------------|--------|---|
| 6 | 1 | First format byte |
| 7 | 1 | Total number of chunks |
| 8 | 1 | Reference number of this chunk (0 to total number of chunks - 1) |
| 9 | 1 | For all chunks: Size of this chunk (in useful data bytes, so ignoring the '8 th bit' bytes) |

The number of data bytes remaining is a count of bytes excluding the '8th bit' bytes and is present in each chunk. In all full chunks this is 0x19.

Figure 13 shows the overall structure of a multi-chunk message received from the amp.

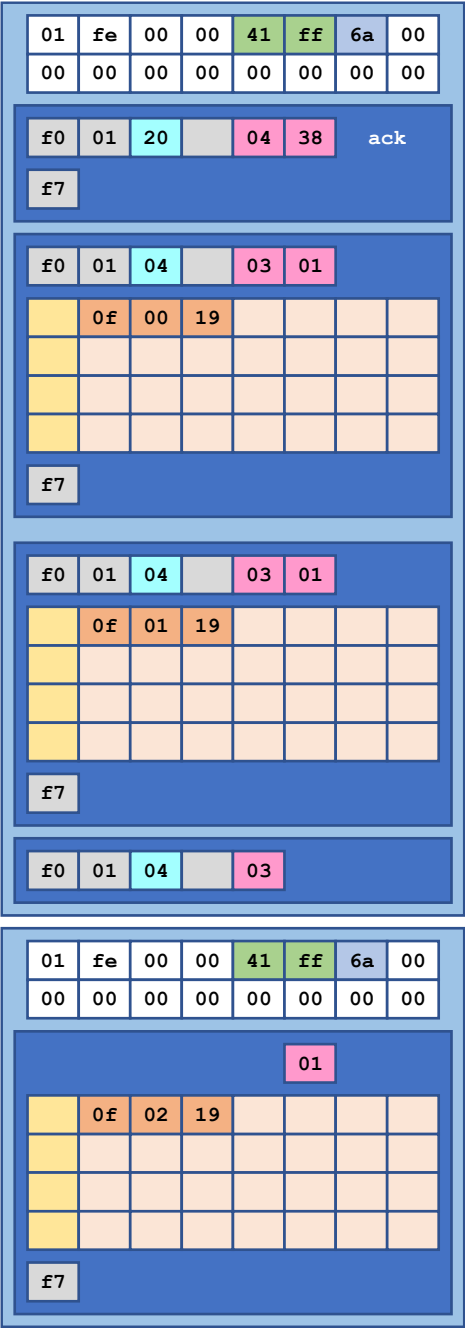


Figure 13

Commands sent to the amp

These are the commands which can be sent to the amp and the responses expected.

| Command | Sub-command | Meaning | Response | Model |
|---------|-------------|----------------------------------|---------------------|-------|
| 01 | 01 | Send preset details to the amp | Acknowledge message | All |
| 01 | 04 | Send new effect parameter | None | All |
| 01 | 06 | Change effect to new effect | Acknowledge message | All |
| 01 | 15 | Enable / disable an effect | Acknowledge message | All |
| 01 | 28 | Enable / disable amp volume info | Acknowledge message | All |
| 01 | 33 | Mixer adjustment | Acknowledge message | LIVE |
| 01 | 38 | Change to a different preset | Acknowledge message | All |
| 01 | 62 | Tap tempo | | All |
| 01 | 65 | Tuner on/off | | All |
| 01 | 70 | Send license key | Acknowledge message | All |
| 01 | 72 | Set power settings | Acknowledge message | LIVE |
| 01 | 74 | Change input impedance | Acknowledge message | LIVE |

| Command | Sub-command | Meaning | Response | Model |
|---------|-------------|--------------------------------------|---------------------------|------------|
| 02 | 01 | Get preset details from amp | Preset information | All |
| 02 | 10 | Get current hardware preset number | Preset number information | 40 GO MINI |
| 02 | 11 | Get amp name ("Spark 40") | Amp name | All |
| 02 | 23 | Get amp serial number | Amp serial number | All |
| 02 | 24 | UNKNOWN | | |
| 02 | 2a | Get hardware preset stored checksums | Four checksums | 40 GO MINI |
| 02 | 2b | Get LIVE checksums | Eight checksums | LIVE |
| 02 | 2f | Get firmware version number | Firmware version number | All |
| 02 | 33 | Get mixer setting | Mixer value | LIVE |
| 02 | 71 | | | |
| 02 | 72 | Get power settings | Power settings | LIVE |
| 02 | 74 | Get input impedance | Impedance | LIVE |

Commands sent from the amp

These are the commands / responses sent from the amp. Response to the amp are unknown.

| Command | Sub-command | Meaning | Response | Model |
|---------|-------------|---|----------|------------|
| 03 | 01 | Response to a preset information query command | None | All |
| 03 | 06 | Change of effect (amp model) on the amp | None | All |
| 03 | 10 | Current hardware preset on amp | None | All |
| 03 | 11 | Amp name | None | All |
| 03 | 15 | Enable / disable an effect | None | All |
| 03 | 1a | Store preset on amp OR Response to 0x021a request | None | LIVE |
| 03 | 23 | Amp serial number | None | All |
| 03 | 27 | Store current preset in hardware preset | None | All |
| 03 | 28 | Send amp info (currently volume for the JH wah) | None | All |
| 03 | 2a | Stored hardware preset checksums | None | 40 GO MINI |
| 03 | 2b | Stored hardware preset checksums | | LIVE |
| 03 | 2f | Firmware version number | None | All |
| 03 | 33 | Value of mixer | None | LIVE |
| 03 | 37 | Change of effect parameter on amp | None | All |
| 03 | 38 | Change of preset selected on the amp | None | 40 GO MINI |
| 03 | 63 | Tap tempo | None | All |
| 03 | 64 | Tuner On | None | All |
| 03 | 65 | Tuner Off | None | All |
| 03 | 6b | Input 1 GUITAR volume | None | LIVE |

| | | | | |
|----|----|----------------------|------|------|
| 03 | 71 | | | |
| 03 | 72 | Power settings | None | LIVE |
| 03 | 73 | Input 2 cable insert | None | LIVE |
| 03 | 74 | Impedance response | None | LIVE |

| Command | Sub-command | Meaning | Response | Model |
|---------|--------------------------------|--|----------|-------|
| 04 | As per command received by amp | Acknowledgement from the amp that it received a message. | | All |
| 04 | 28 | | | |
| 04 | 70 | | | |
| 04 | 72 | | | |

Detail of commands

0x0101 - see later

0x0104 - change effect parameter

| Type | Length | Content | Example |
|--------------------------|--------|---|-----------------------------|
| Alternative short string | n+2 | Effect name | 0x04 0xa4 Twin |
| Integer | 1 | Number of the parameter starting at 0 | 0x00 (Gain) |
| Float | 5 | Value for the parameter (0-1.0, with 1.0 representing 10 in the user interface) | 0xca 0x3f 0x21 0x72 0x13 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x00 |

0x0106 - swap effects

| Type | Length | Content | Example |
|--------------------------|--------|-------------------------------------|--------------------|
| Alternative short string | n+2 | Old effect name | 0x08 0xa8 LA2AComp |
| Alternative short string | n+2 | New effect name | 0x08 0xa8 BlueComp |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x00 |

0x0115 - enable / disable effect

| Type | Length | Content | Example |
|--------------------------|--------|-------------------------------------|--------------------|
| Alternative short string | n+2 | New effect name | 0x08 0xa8 BlueComp |
| Boolean | 1 | New status 0xc2 off 0xc3 on | 0xc3 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x00 |

0x0128 - enable / disable amp info

So far only seen on JH.Vox846 wah - asks the amp to send volume change information back to the app

| Type | Length | Content | Example |
|--------------------------|--------|-----------------------------------|---------------------|
| Alternative short string | n+2 | Effect name | 0x09 0xa9 JH.Wah846 |
| Boolean | 1 | New status 0xc2 off 0xc3 on | 0xc3 |

0x0138 - change to a new hardware preset

| Type | Length | Content | Example |
|---------|--------|--------------------------------|---------|
| Integer | 1 | 0 | 0x00 |
| Integer | 1 | New preset number 0-3, 0x7f | 0x03 |

0x0162 - send tap tempo

| Type | Length | Content | Example |
|-------|--------|-------------|-----------------------------|
| Float | 5 | Tempo value | 0xca 0x3f 0x21 0x72 0x13 |

0x0165 - tuner on/off

| Type | Length | Content | Example |
|------|--------|-----------------|---------|
| Bool | 1 | Tuner on or off | 0xc2 |

0x0170 - send license key

| Type | Length | Content | Example |
|---------|--------|---------------------|---------|
| Integer | 64 | 64 byte license key | |

0x0172 - set power settings

| Type | Length | Content | Example |
|---------|--------|---|---------|
| Boolean | 1 | | 0xc2 |
| Integer | 1 | Auto-shutdown time 0 = Never 30 40 50 70 | 0x00 |
| Integer | 1 | | |
| Integer | 1 | Auto standby time 0 = Never 5 10 15 30 | 0x05 |

0x0174 - set input impedance

| Type | Length | Content | Example |
|-------------|--------|---|---------|
| Fixed array | 1 | Array size | 0x91 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 | 0x01 |
| Integer | 1 | Impedance 0 = Standard 1 = Hi-Z 2 = Line 3 = Mic | 0x00 |

0x0133 - change mixer setting

| Type | Length | Content | Example |
|---------|--------|--|-----------------------------|
| Integer | 1 | Mixer channel 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 5 = Music 9 = Master | 0x03 |
| Float | 5 | Value | 0xca 0x3f 0x21 0x72 0x13 |

0x0201 - get preset information

| Type | Length | Content | Example |
|---------------|--------|--|-----------|
| Integer | 2 | hardware preset 0x00-0x03 software preset 0x7f current live preset 0x0100 | 0x00 0x03 |
| Short integer | 1 x 30 | 30 bytes of 0x00 (which, when adding the 8 th bit in a separate byte, looks like 34 bytes of 00 in the raw data) | 0x00 |

0x0210 - get hardware preset number

No data in message for this command

Response is a single string.

0x0211 - get amp name

No data in message for this command

Response is a single string.

0x021a - get hardware preset information (LIVE)

| Type | Length | Content | Example |
|-------------|--------|-------------------------------------|---------|
| Fixed array | 1 | 2 | 0x92 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | |

Retrieves the checksums for the hardware presets.

Response is an array of eight integers.

0x0223 - get amp serial number

No data in message for this command

Response is a single string

0x0224 - UNKNOWN

| Type | Length | Content | Example |
|---------|--------|---------|---------|
| Integer | 1 | | 0x00 |
| Integer | 1 | | 0x01 |
| Integer | 1 | | 0x02 |
| Integer | 1 | | 0x03 |

0x022a - get hardware preset checksums

Retrieves the checksums for the hardware presets.

Response is an array of 4 integers.

0x022f - get amp firmware version number

No data in message for this command

Response is a msgpack uint32 - 4 byte int.

0x022b - get hardware preset checksums (LIVE)

| Type | Length | Content | Example |
|---------|--------|-------------------------------------|---------|
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x01 |

Retrieves the checksums for the hardware presets.

Response is an array of eight integers.

0x0233 - get mixer setting (LIVE)

| Type | Length | Content | Example |
|---------|--------|--|---------|
| Integer | 1 | Mixer channel 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 5 = Music 9 = Master | 0x01 |

0x0272 - request power settings

No data in message for this command

0x0273 - request input status

| Type | Length | Content | Example |
|-------------|--------|---|---------|
| Fixed array | 1 | Size of array | 0x92 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 | 0x00 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 | 0x01 |

0x0274 - request input impedance

| Type | Length | Content | Example |
|-------------|--------|---|---------|
| Fixed array | 1 | Size of array | 0x91 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 | 0x00 |

0x0306 - change of effect (amp model) on the amp

| Type | Length | Content | Example |
|--------------------------|--------|-------------------------------------|--------------------|
| Alternative short string | n+2 | Old amp name | 0x0d 0xad GK800 |
| Alternative short string | n+2 | New amp name | 0x05 0xa5 Twin |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x00 |

When this is sent to the app, the app responds by sending five parameter messages back to the amp (0x0104). These cover the five parameters (gain, bass, middle, treble and volume). It is not clear how these parameter values are derived.

0x0310 - current hardware preset

| Type | Length | Content | Example |
|---------|--------|--------------------------------|---------|
| Integer | 1 | New preset current | 0x00 |
| Integer | 1 | New preset number 0-3, 0x7f | 0x03 |

0x0311 - amp name

| Type | Length | Content | Example |
|--------------------------|--------|----------|--------------------|
| Alternative short string | n+2 | Amp name | 0x08 0xa8 Spark 40 |

0x0315 - enable / disable effect

| Type | Length | Content | Example |
|--------------------------|--------|-----------------------------------|--------------------|
| Alternative short string | n+2 | New effect name | 0x08 0xa8 BlueComp |
| Boolean | 1 | New status 0xc2 off 0xc3 on | 0xc3 |

Only seems to come from the amp when using the Mod or Delay knobs.

0x031a - response to 0x021a request

| Type | Length | Content | Example |
|-------------|--------|--|---------|
| Fixed array | 1 | Size of array | 0x92 |
| Integer | 1 | Current preset input 1 | 0x00 |
| Integer | 1 | Preset number input 1 | 0x05 |
| Boolean | 1 | Content changed 0xc2 hasn't changed 0xc3 has changed | 0xc3 |
| Integer | 1 | Current preset input 2 | 0x00 |
| Integer | 1 | Preset number input 2 | 0x01 |
| Boolean | 1 | Content changed 0xc2 not changed 0xc3 has changed | 0xc2 |

0x031a - store preset on amp

| Type | Length | Content | Example |
|-------------|--------|-------------------|---------|
| Fixed array | 1 | Size of array | 0x91 |
| Integer | 1 | Current preset | 0x00 |
| Integer | 1 | Preset number | 0x05 |
| Boolean | 1 | Unchanged 0xc2 | 0xc2 |

0x0323 - amp serial number

| Type | Length | Content | Example |
|--------------------------|--------|-------------------|-------------------------|
| Alternative short string | n+2 | Amp serial number | 0x10 0x30 S1011G1222 |

0x0327 - current preset stored to hardware on the amp

| Type | Length | Content | Example |
|---------|--------|--|------------|
| Integer | 1 | 0 | 0x00 |
| Integer | 1 | Number of the preset where settings are stored | x02 (2) |

0x0328 - amp effect level

| Type | Length | Content | Example |
|-------|--------|--|-----------------------------|
| Float | 5 | Value for the parameter (0-1.0, with 1.0 representing 10 in the user interface) | 0xca 0x3f 0x21 0x72 0x13 |

Only seems to come from the amp when using the Mod or Delay knobs.

0x032a - stored presets checksum (40 / GO / MINI)

| Type | Length | Content | Example |
|-------------|--------|---------|-------------|
| Fixed array | 1 | 0x98 | 0x98 (8) |
| Integer | 1 or 2 | 0x00 | 0x00 |
| Integer | 1 or 2 | 0x01 | 0x01 |
| Integer | 1 or 2 | 0x02 | 0x02 |
| Integer | 1 or 2 | 0x03 | 0x03 |

0x032b - stored preset checksum (LIVE)

| Type | Length | Content | Example |
|-------------|--------|---------|-------------|
| Fixed array | 1 | 0x98 | 0x98 (8) |
| Integer | 1 or 2 | 0x00 | 0x00 |
| Integer | 1 or 2 | 0x01 | 0x01 |
| Integer | 1 or 2 | 0x02 | 0x02 |
| Integer | 1 or 2 | 0x03 | 0x03 |
| Integer | 1 or 2 | 0x00 | 0x00 |
| Integer | 1 or 2 | 0x01 | 0x01 |
| Integer | 1 or 2 | 0x02 | 0x02 |
| Integer | 1 or 2 | 0x03 | 0x03 |

0x0333 - LIVE mixer setting

| Type | Length | Content | Example |
|-------|--------|------------------|-----------------------------|
| Float | 5 | Volume for input | 0xca 0x3e 0x6d 0x5b 0x37 |

0x0337 - change of parameter for effect on the amp

| Type | Length | Content | Example |
|--------------------------|--------|-------------------------------------|--|
| Alternative short string | n+2 | Effect name | 0x04 0x24 Twin |
| Integer | 1 | Parameter number | 0x00 (0) (Gain) OR 0x03 (3) (Bass) OR 0x04 (4) (Master) |
| Float | 5 | New value | 0xca 0x3e 0x6d 0x5b 0x37 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 | 0x00 |

0x0338 - change of preset selected on the amp

| Type | Length | Content | Example |
|---------|--------|--------------------------|-------------|
| Integer | 1 | 0 | 0x00 |
| Integer | 2 | Number of the new preset | 0x02 (2) |

0x0363 - tap tempo

| Type | Length | Content | Example |
|---------|--------|-------------|-----------------------------|
| Float | 5 | Tempo value | 0xca 0x3f 0x21 0x72 0x13 |
| Integer | 1 | | 0x3f |
| Integer | 1 | | 0x3f |

0x036b - Input 1 GUITAR volume

| Type | Length | Content | Example |
|-------|--------|---------|-----------------------------|
| Float | 5 | Volume | 0xca 0x3e 0x6d 0x5b 0x37 |

0x0372 - power settings

| Type | Length | Content | Example |
|---------|--------|---|---------|
| Boolean | 1 | | 0xc2 |
| Integer | 1 | Auto-shutdown time 0 = Never 30 40 50 70 | 0x00 |
| Integer | 1 | | |
| Integer | 1 | Auto standby time 0 = Never 5 10 15 30 | 0x05 |

0x0373 - Input 2 cable insert

| Type | Length | Content | Example |
|-------------|--------|--|---------|
| Fixed array | 1 | Size of array | 0x91 |
| Integer | 1 | Input 1 = Input 2 | 0x01 |
| Integer | 1 | Plug type 1 = 1/4" jack 2 = XLR | 0x01 |
| Boolean | 1 | Plug status 0xc2 unplugged 0xc3 plugged in | 0xc3 |

0x0374 - Impedance response

| Type | Length | Content | Example |
|-------------|--------|---|---------|
| Fixed array | 1 | Size of array | 0x91 |
| Integer | 1 | Input 0 = Input 1 1 = Input 2 1/4" 2 = Input 2 XLR 3 = Input 3 4 = Input 4 | 0x01 |
| Integer | 1 | Impedance 0 = Standard 1 = Hi-Z 2 = Line 3 = Mic | 0x00 |

0x04nn / 0x05nn - acknowledgement

This has command 0x04 and the same sub-command as was issued to the amp. It has the same sequence number as the command issued to the app.

The possible acknowledgements are:

| Type | Length | Example |
|----------------------|--------|------------|
| Preset chunk | 0 | 0x04 01 |
| Preset final chunk | 0 | 0x05 01 |
| Change amp model | 0 | 0x04 06 |
| Turn on/off | 0 | 0x04 15 |
| Turn amp info on/off | 1 | 0x04 28 c2 |
| Change preset number | 0 | 0x04 38 |
| License key | 1 | 0x04 70 xx |
| Power settings | 1 | 0x04 72 00 |

For 70, 28 and 72 there is a 1 byte body to the message, for the rest they are empty - just the chunk header and the trailer (0xf7)

0x0101 - send preset

A new preset is a multi-chunk message, so the first three bytes of each new chunk are the chunk sub-header.

The preset format contains data for the preset, and then information for each effect - 7 in total.

Each effect contains data for the effect, and then a value for each parameter in the effect.

The final byte of the preset is a checksum.

| Type | Length | Content | Example |
|----------------------------|--------------|--|---|
| Integer | 1 | Current indicator | 0x00 |
| Integer | 1 | Hardware preset 0-3 Software preset 0x7f | 0x7f |
| Long string | 36 | UUID of preset | |
| Short string | n+1 | Name | 0xad Spooky Melody |
| Short string | n+1 | Version | 0xa3 0.7 |
| Short string / Long string | n+1 / n+2 | Description | 0xb7 Description for Alternative Preset 1 |
| Short string | n+1 | Icon name | 0xa8 icon.png |
| Float | 5 | BPM | 0xca 0x42 0x70 0x00 0x00 (60.0) |
| Fixed array | 1 | Number of effects | 0x97 (7) |
| Effect 0 | | See below | |
| Parameter 0 | | See below | |
| Parameter 1 | | See below | |
| Effect 1 | | See below | |
| Parameter 0 | | See below | |
| Effect 2 | | See below | |
| Effect 3 | | See below | |
| Effect 4 | | See below | |
| Effect 5 | | See below | |
| Effect 6 | | See below | |
| Integer | 1 | Checksum of the msgpack formatted data - excluding preset location (first two bytes) Sum of all bytes modulo 256. | |

Each effect then has a section describing the effect (7 effects in total)

| Type | Length | Content | Example |
|--------------|--------|---|--------------------|
| Short string | n+1 | Effect name | 0x08 0xa8 BlueComp |
| Boolean | 1 | Status 0xc2 off 0xc3 on | 0xc3 (On) |
| Fixed array | 1 | Number of parameters for this effect | 0x94 (4) |

And then each parameter has a section describing the value for the parameter

| Type | Length | Content | Example |
|-------------|--------|--------------------------------|-----------------------------|
| Integer | 1 | Parameter reference | 0x01 (1) |
| Fixed array | 1 | Number of values (always 1) | 0x91 |
| Float | 5 | Value for this parameter | 0xca 0x3e 0x35 0x55 0x3f |

It seems this is a broken implementation of msgpack, because the fixed array for each parameter looks more like it should be key:value pairs, and the fixed array wrapping the float seems redundant.

What should be:

```
fixmap(3) = {{1, float}, {2, float}, {3, float}}
```

is actually:

```
fixarray(3)=[0, fixarray(1)[float], 1], fixarray(1)[float], 2, fixarray(1)[float]
```

Parsing can be achieved by ignoring any integer within a fixarray and by extracting the float from within a fixarray of length 1 - but this is a workaround for a broken implementation.

It seems the amp and app do not pack or unpack msgpack properly - they must be looking for the specific data rather than unpacking and indexing the unpacked data

Figure 14 shows the overall structure.

| | | | | | | | | | | | | | | |
|------------------------------|-----|----|---|---|---|---|---|---|---|---|---|---|---|---|
| Hardware preset number | 00 | 7f | | | | | | | | | | | | |
| UUID | 0 | 7 | 0 | 7 | 9 | 0 | 6 | 3 | - | 9 | 4 | A | 9 | - |
| | 4 | 1 | B | 1 | - | A | B | 1 | D | - | 0 | 2 | C | B |
| | 5 | D | 0 | 0 | 7 | 9 | 0 | | | | | | | |
| Name | D | a | r | k | | S | o | u | l | | | | | |
| Version | 0 | . | 7 | | | | | | | | | | | |
| Description | 1 | - | C | l | e | a | n | | | | | | | |
| Icon | i | c | o | n | . | p | n | g | | | | | | |
| BPM | 120 | | | | | | | | | | | | | |
| Number of effects (fixarray) | 97 | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | |
|---------------------------------|----|----|------|---|---|---|---|---|---|---|---|---|---|---|
| Effect name | b | i | a | s | . | n | o | i | s | e | g | a | t | e |
| On / off | C3 | | | | | | | | | | | | | |
| Number of parameters (fixarray) | 93 | | | | | | | | | | | | | |
| Parameter 0 | 00 | 91 | 0.52 | | | | | | | | | | | |
| Parameter 1 | 01 | 91 | 0.88 | | | | | | | | | | | |
| Parameter 2 | 02 | 91 | 0.13 | | | | | | | | | | | |

.....

| | | | | | | | | | | | | | | |
|---------------------------------|-----|-----|------|-----|-----|-----|---|---|---|---|---|--|--|--|
| Effect name | b | i | a | s | . | r | e | v | e | r | b | | | |
| On / off | C3 | | | | | | | | | | | | | |
| Number of parameters (fixarray) | 97 | | | | | | | | | | | | | |
| Parameter 0 | 00 | 91 | 0.52 | | | | | | | | | | | |
| ... | ... | ... | ... | ... | ... | ... | | | | | | | | |
| Parameter 6 | 06 | 91 | 0.13 | | | | | | | | | | | |

| | | | | | | | | | | | | | | |
|----------|----|--|--|--|--|--|--|--|--|--|--|--|--|--|
| Checksum | 17 | | | | | | | | | | | | | |
|----------|----|--|--|--|--|--|--|--|--|--|--|--|--|--|

Figure 14

Appendix 1 - Preset locations

The amp has 4 presets which can be selected from the top panel. These are hardware presets represented by presets 0x00-0x03 in this document.

When the app sends one of the 'non-hardware' presets this is sent as preset 0x7f.

Preset 0x7f cannot be selected from the top panel.

Preset data is static within the amp, but multiple parameters can be modified either via the amp top panel (amp type, gain, treble, modulation etc) or via the app. When they have been modified the preset led flashes to show the amp state is currently different from the stored preset.

This is the **current state** of the amp - it does not necessarily map to any of the presets because the state has been modified.

It may help to think of the presets as static data sent to the amp and stored. A preset can be selected to make it the **current state**, and changes made will change the **current state**. This could then be saved back to a preset location. But it requires the 'save' - changes made on the app or the amp do not automatically change the preset.

The presets are only updated when the current state is stored into the current preset (on the app), a new named preset (on the app) or via a long press of a hardware preset button (on the amp). This will cause the led to stop flashing.

It is possible to create a preset and send it to any location in the amp - 0x00-0x03, 0x7f - just use the 'send new preset' command with the preset location between 0 and 3.

Sending preset details does not enable that preset - a separate command is needed to move the amp to that preset.

It is possible to retrieve preset details from the amp.

Reading preset 0x0000-0x0003, 0x007f gives the **stored state** of that preset.

Reading preset 0x0100 gives the current effects in use - so **amp current state**.

Sending a preset only uses the low byte as 0-3, 0x7f.

Receiving a preset uses the low byte UNLESS the high byte is 0x01, in which case the current state is retrieved.

If high byte is 0x01 in the retrieved data the low byte should be ignored.

Hardware preset location

Software preset location

| | | | |
|-------|-------|-------|-------|
| 00 00 | 00 01 | 00 02 | 00 03 |
|-------|-------|-------|-------|

| |
|-------|
| 00 7f |
|-------|

| |
|------|
| 01xx |
|------|

Appendix 2 - Effect and amp names

Noisegate

| Name | Spark name |
|-----------|----------------|
| Noisegate | bias.noisegate |

Compressors

| Name | Spark name |
|--------------------|----------------|
| LA Comp | LA2AComp |
| Sustain Comp | BlueComp |
| Red Comp | Compressor |
| Bass Comp | BassComp |
| Optical Comp | BBEOpticalComp |
| J.H. Legendary Wah | JH.Vox846 |

Drive

| Name | Spark name |
|------------------|-------------------|
| Booster | Booster |
| Tube Drive | DistortionTS9 |
| Over Drive | Overdrive |
| Fuzz Face | Fuzz |
| Black Op | ProCoRat |
| Bass Muff | BassBigMuff |
| Guitar Muff | GuitarMuff |
| Bassmaster | MaestroBassmaster |
| SAB Driver | SABdriver |
| Clone Drive | KlonCentaurSilver |
| J.H. Axle Fuzz | JH.AxisFuzz |
| J.H. Super Fuzz | JH.SupaFuzz |
| J.H. Octave Fuzz | JH.Octavia |
| J.H. Fuzz Tone | JH.FuzzTone |

Amps

| Name | Spark name |
|--------------------|-------------------|
| Silver 120 | RolandJC120 |
| Black Duo | Twin |
| AD Clean | ADClean |
| Match DC | 94MatchDCV2 |
| Tweed Bass | Bassman |
| AC Boost | AC Boost |
| Checkmate | Checkmate |
| Two Stone SP50 | TwoStoneSP50 |
| American Deluxe | Deluxe65 |
| Plexiglass | Plexi |
| JM45 | OverDrivenJM45 |
| Lux Verb | OverDrivenLuxVerb |
| RB 101 | Bogner |
| British 30 | OrangeAD30 |
| American High Gain | AmericanHighGain |
| SLO 100 | SLO100 |
| YJM100 | YJM100 |
| Treadplate | Rectifier |
| Insane | EVH |
| Switch Axe | SwitchAxeLead |
| Rocker V | Invader |
| BE 101 | BE101 |

| | |
|--------------------|--------------------|
| Pure Acoustic | Acoustic |
| Fishboy | AcousticAmpV2 |
| Jumbo | FatAcousticV2 |
| Flat Acoustic | FlatAcoustic |
| RB-800 | GK800 |
| Sunny 3000 | Sunny3000 |
| W600 | W600 |
| Hammer 500 | Hammer500 |
| ODS 50 | ODS50CN |
| J.h. D-Show Master | JH.DualShowman |
| J.H. Sun 100S | JH.Sunn100 |
| Blues Boy | BluesJrTweed |
| J.H. 45/100 | JH.JTM45 |
| J.H. Bass Master | JH.Bassman50Silver |
| J.H. Super 100 | JH.SuperLead100 |
| J.H. Tone City 100 | JH.SoundCity100 |
| Insane 6508 | 6505Plus |

Modulation

| Name | Spark name |
|-------------------|-----------------|
| Tremolo | Tremolo |
| Chorus | ChorusAnalog |
| Flanger | Flanger |
| Phaser | Phaser |
| Vibrato | Vibrato01 |
| UniVibe | UniVibe |
| Cloner Chorus | Cloner |
| Classic Vibe | MiniVibe |
| Tremolator | Tremolator |
| Tremolo Square | TremoloSquare |
| JH Legendary Vibe | JH.VoodooVibeJr |
| Guitar QA | GuitarEQ6 |
| Bass EQ | BassEQ6 |

Delay

| Name | Spark name |
|---------------|----------------|
| Digital Delay | DelayMono |
| Echo Filt | DelayEchoFilt |
| Vintage Delay | VintageDelay |
| Reverse Delay | DelayReverse |
| Multi Head | DelayMultiHead |
| Echo Tape | DelayRe201 |

Reverb

| Name | Spark name |
|-------------|-------------|
| All Reverbs | bias.reverb |

Appendix 3 - app startup messages

These are the messages sent when the app connects to the Spark amp.

Sent to amp

| Command | Parameter | Description |
|---------|-------------|------------------------------------|
| 02 11 | | Get amp name |
| 02 2a | 00 01 02 03 | Unknown |
| 02 23 | | Get serial number |
| 01 70 | | Send license key |
| 02 01 | 00 00 | Get preset 0 |
| 02 01 | 00 01 | Get preset 1 |
| 02 01 | 00 02 | Get preset 2 |
| 02 01 | 00 03 | Get preset 3 |
| 02 10 | | Get current hardware preset on amp |
| 02 2F | | Get firmware version |
| 02 01 | 01 00 | Get current amp settings |

Response from amp

| Command | Example | Description |
|---------|-----------------------------|---------------------------------------|
| 03 11 | 0x08 Spark 40 | Amp name |
| 03 2a | 0x94 0x4c 0x56 0x67 0x9c | Preset checksums |
| 03 23 | Serial number 0xf7 | Serial number |
| 03 01 | | Preset 0 |
| 03 01 | | Preset 1 |
| 03 01 | | Preset 2 |
| 03 01 | | Preset 3 |
| 03 10 | 00 01 | Hardware preset number |
| 03 2F | 01 00 02 fd | Firmware version |
| 03 01 | | Current amp settings in preset format |

Appendix 4 – Calculating effective data bytes from total number of bytes including format byte

This visualises how to calculate the number of data bytes to go into the multi-chunk sub-header:

$$\text{total_bytes} - \text{int} \left(\frac{\text{total_bytes} + 2}{8} \right)$$

| | bytes | bytes+2 | int((bytes+2) / 8) | bytes – int((bytes+2) / 8) | | | | | | | | | | | | | | | | | | | | | | | | |
|---|-------|---------|---------------------|-----------------------------|----|----|----|----|----|---|----|----|----|----|--|--|--|--|--|--|--|--|--|--|---|----|---|---|
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>f7</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | f7 | | | | | | | | | | | | | | | | | | | 1 | 3 | 0 | 1 |
| | | | | n | f7 | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>f7</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | f7 | | | | | | | | | | | | | | | | | | 2 | 4 | 0 | 2 |
| | | | | n | n | f7 | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>f7</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | f7 | | | | | | | | | | | | | | | | | 3 | 5 | 0 | 3 |
| | | | | n | n | n | f7 | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>f7</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | f7 | | | | | | | | | | | | | | | | 4 | 6 | 0 | 4 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| f7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>f7</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | f7 | | | | | | | | | | | | | | 6 | 8 | 1 | 5 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | f7 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>f7</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | f7 | | | | | | | | | | | | | 7 | 9 | 1 | 6 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | f7 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>f7</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | f7 | | | | | | | | | | | | 8 | 10 | 1 | 7 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | f7 | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>n</td><td>f7</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | n | f7 | | | | | | | | | | | 9 | 11 | 1 | 8 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | n | f7 | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|---|----|---|---|---|----|----|---|---|---|---|---|---|---|----|----|----|---|----|--|--|--|--|--|----|----|---|----|
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>f7</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | n | n | f7 | | | | | | | | | | 10 | 12 | 1 | 9 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | n | n | f7 | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>f7</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | n | n | n | f7 | | | | | | | | | 11 | 13 | 1 | 10 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | n | n | n | f7 | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>f7</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | n | n | n | n | f7 | | | | | | | | 12 | 14 | 1 | 11 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| f7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td></td><td></td><td></td><td></td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td><td>n</td></tr><tr><td>x</td><td>n</td><td>f7</td><td></td><td></td><td></td><td></td><td></td></tr></table> | | | | | n | n | n | n | x | n | n | n | n | n | n | n | x | n | f7 | | | | | | 14 | 16 | 2 | 12 |
| | | | | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | n | n | n | n | n | n | | | | | | | | | | | | | | | | | | | | | |
| x | n | f7 | | | | | | | | | | | | | | | | | | | | | | | | | | |

Appendix 5 - SysEx 7bit/8bit encoding functions

The best reference I could find is in the Arduino MIDI library and refers to Ruin & Wesen's SysEx encoder/decoder - <http://ruinwesen.com>

Sadly this reference is broken so there is no clarity on whether this encoding has a recognised name.

Ian McKellar

<https://git.sr.ht/~ianloic/spark-usb-midi>

```
def decode_block(block: bytes) -> bytes:
    assert(len(block) > 0)
    top = block[0]
    bottom = block[1:]
    assert(len(bottom) <= 7)
    decoded = []
    for i, b in enumerate(bottom):
        if top & (2**i):
            decoded.append(b | 2**7)
        else:
            decoded.append(b)
    return bytes(decoded)
```

My code

<https://github.com/paulhamsh/Spark-Parser/blob/main/MidiControl/SparkClass.py>

```
chunk_len = len(chunk)
num_seq = int((chunk_len + 6) / 7)
bytes7 = b''

for this_seq in range(0, num_seq):
    seq_len = min(7, chunk_len - (this_seq * 7))
    bit8 = 0
    seq = b''
    for ind in range(0, seq_len):
        dat = chunk[this_seq * 7 + ind]
        if dat & 0x80 == 0x80:
            bit8 |= (1 << ind)
        dat &= 0x7f
        seq += bytes([dat])
    bytes7 += bytes([bit8]) + seq

chunk_len = len(data7bit)
num_seq = int((chunk_len + 7) / 8)
data8bit = b''
for this_seq in range(0, num_seq):
    seq_len = min(8, chunk_len - (this_seq * 8))
    seq = b''
    bit8 = data7bit[this_seq * 8]
    for ind in range(0, seq_len-1):
        dat = data7bit[this_seq * 8 + ind + 1]
        if bit8 & (1 << ind) == (1 << ind):
            dat |= 0x80
        seq += bytes([dat])
    data8bit += seq
```


Arduino MIDI

https://github.com/FortySevenEffects/arduino_midi_library/blob/master/src/MIDI.cpp

```
/*! \brief Encode System Exclusive messages.
SysEx messages are encoded to guarantee transmission of data bytes higher than
127 without breaking the MIDI protocol. Use this static method to convert the
data you want to send.
\param inData The data to encode.
\param outSysEx The output buffer where to store the encoded message.
\param inLength The length of the input buffer.
\param inFlipHeaderBits True for Korg and other who store MSB in reverse order
\return The length of the encoded output buffer.
@see decodeSysEx
Code inspired from Ruin & Wesen's SysEx encoder/decoder - http://ruinwesen.com
*/
unsigned encodeSysEx(const byte* inData,
                    byte* outSysEx,
                    unsigned inLength,
                    bool inFlipHeaderBits)
{
    unsigned outLength = 0;    // Num bytes in output array.
    byte count = 0;           // Num 7bytes in a block.
    outSysEx[0] = 0;

    for (unsigned i = 0; i < inLength; ++i)
    {
        const byte data = inData[i];
        const byte msb = data >> 7;
        const byte body = data & 0x7f;

        outSysEx[0] |= (msb << (inFlipHeaderBits ? count : (6 - count)));
        outSysEx[1 + count] = body;

        if (count++ == 6)
        {
            outSysEx += 8;
            outLength += 8;
            outSysEx[0] = 0;
            count = 0;
        }
    }
    return outLength + count + (count != 0 ? 1 : 0);
}

/*! \brief Decode System Exclusive messages.
SysEx messages are encoded to guarantee transmission of data bytes higher than
127 without breaking the MIDI protocol. Use this static method to reassemble
your received message.
\param inSysEx The SysEx data received from MIDI in.
\param outData The output buffer where to store the decrypted message.
\param inLength The length of the input buffer.
\param inFlipHeaderBits True for Korg and other who store MSB in reverse order
\return The length of the output buffer.
@see encodeSysEx @see getSysExArrayLength
Code inspired from Ruin & Wesen's SysEx encoder/decoder - http://ruinwesen.com
*/
unsigned decodeSysEx(const byte* inSysEx,
                    byte* outData,
                    unsigned inLength,
                    bool inFlipHeaderBits)
{
    unsigned count = 0;
    byte msbStorage = 0;
    byte byteIndex = 0;

    for (unsigned i = 0; i < inLength; ++i)
    {
```

```
    if ((i % 8) == 0)
    {
        msbStorage = inSysEx[i];
        byteIndex = 6;
    }
    else
    {
        const byte body      = inSysEx[i];
        const byte shift     = inFlipHeaderBits ? 6 - byteIndex : byteIndex;
        const byte msb       = byte(((msbStorage >> shift) & 1) << 7);
        byteIndex--;
        outData[count++] = msb | body;
    }
}
return count;
}
```

Appendix 6 - msgpack

This format is described at:

www.msgpack.org

<https://github.com/msgpack/msgpack/blob/master/spec.md>

There are multiple implementations and the python one is obtained by:

```
python -m pip install msgpack
```

The Spark data is not an exact msgpack implementation because it does not start as an array, and the effect and effect parameters are malformed as arrays.

The data is like this:

```
\x97
  \xae\bias.noisegate
  \xc2
  \x93
    \x00
    \x91
      \xca>\r\xal\xec
    \x01
    \x91
      \xca>f\x08\xd1
    \x02
    \x91
      \xca\x00\x00\x00\x00
```

This should be an array of 7 elements - one for each effect. Then each effect has three values - name, on/off status, an array of parameters - best as a key/value pair except that msgpack doesn't allow integers as keys.

But the array content is like this:

```
['bias.noisegate',
True,
[0, [0.1201], 1],
[0.3314],
2,
[0.0000]
```

Partly because the array is really three entries per pedal, not one, and partly because each parameter is two entries not one.