Worked example of creating a preset from raw information

The C structure definition:

```
typedef struct  {
  uint8_t  curr_preset;
  uint8_t  preset_num;
  char UUID[STR_LEN];
  char Name[STR_LEN];
  char Version[STR_LEN];
  char Description[STR_LEN];
  char Icon[STR_LEN];
  float BPM;
  struct SparkEffects {
    char EffectName[STR_LEN];
    bool OnOff;
    uint8_t  NumParameters;
    float Parameters[10];
  } effects[7];
  uint8_t chksum;
} SparkPreset;
```

And a completed example:

```
SparkPreset my_preset{
       0x00,
       0x7f,
       "ba6f4218-75b5-402d-8d49-399286db85aa",
       "AC40 Tone",
       "0.7",
       "My own tone",
       "icon.png",
       120.00, {
       {"bias.noisegate", true,3, {0.28, 0.30, 1.00}},
       {"Compressor", true, 2, {0.38, 0.33}},
       {"Booster", true, 1, {0.66}},
       {"AC Boost", true, 5, {0.51, 0.47, 0.46, 0.53, 0.15}},
       {"ChorusAnalog", false, 4, {0.59, 0.35, 0.60, 0.47}},
       {"DelayEchoFilt",  false, 5, {0.08, 0.50, 0.55, 0.37, 1.00}},
       {"bias.reverb", true, 8, {0.50, 0.25, 0.34, 0.49, 0.34, 0.34, 0.30, 1.00}}},
       167 };
```

Msgpack key formats (from msgpack.org)

Format name	First byte (in hex)	Remaining bytes
positive fixint	0x00 - 0x7f	-
fixmap	0x80 - 0x8f	-
fixarray	0x90 - 0x9f	-
fixstr	0xa0 - 0xbf	-
nil	0xc0	-
false	0xc2	-
true	0xc3	-
uint 32	0xce	4 bytes big endian uint (uint 32 stores a 32-bit big-endian unsigned integer)
float 32	0xca	4 bytes big endian float (big-endian IEEE 754 single precision floating point number)
str 8	0xd9	number of bytes bytes of string

The data is encoded in msgpack format. Strings are either of the format 'fixstr' or 'str 8' (in a simple message the strings are also prefixed with their length, before the msgpack header).

In the Spark implementation each float has an integer and 'array of 1'.

Also the set of floats are indicated by an 'array of x' before them – this should probably be a fixmap rather than fixarray.

Create the msgpack data from the C information

0	00
7f	7F
ba6f4218-75b5-	D9 24 62 61 36 66 34 32 31 38 2D 37 35 62 35 2D
402d-8d49-	34 30 32 64 2D 38 64 34 39 2D
399286db85aa	33 39 39 32 38 36 64 62 38 35 61 61
AC40 Tone	A9 41 43 34 30 20 54 6F 6E 65
0.7	A3 30 2E 37
My own tone	AB 4D 79 20 6F 77 6E 20 74 6F 6E 65
icon.png	A8 69 63 6F 6E 2E 70 6E 67
120.00	CA 42 F0 00 00
{array of 7}	97
bias.noisegate	AE 62 69 61 73 2E 6E 6F 69 73 65 67 61 74 65
true	C3
{array of 3}	93
0, {arr 1}, 0.28	00 91 CA 3E 8F 5C 29
1, {arr 1}, 0.30	01 91 CA 3E 99 99 9A
2, {arr 1}, 1.00	02 91 CA 3F 80 00 00
Compressor	AA 43 6F 6D 70 72 65 73 73 6F 72
true	C3
{array of 2}	92
0, {arr 1}, 0.38	00 91 CA 3E C2 8F 5C
1, {arr 1}, 0.33	01 91 CA 3E A8 F5 C3
Booster	A7 42 6F 6F 73 74 65 72
true	C3
{array of 1}	91
0, {arr 1}, 0.66	00 91 CA 3F 28 F5 C3
AC Boost	A8 41 43 20 42 6F 6F 73 74
true	C3
{array of 5}	95
0, {arr 1}, 0.51	00 91 CA 3F 02 8F 5C
1, {arr 1}, 0.47	01 91 CA 3E F0 A3 D7
2, {arr 1}, 0.46	02 91 CA 3E EB 85 1F
3, {arr 1}, 0.53	03 91 CA 3F 07 AE 14
4, {arr 1}, 0.15	04 91 CA 3E 19 99 9A
ChorusAnalog	AC 43 68 6F 72 75 73 41 6E 61 6C 6F 67
false	C2
{array of 4}	94
0, {arr 1}, 0.59	00 91 CA 3F 17 0A 3D
1, {arr 1}, 0.35	01 91 CA 3E B3 33 33
2, {arr 1}, 0.60	02 91 CA 3F 19 99 9A
3, {arr 1}, 0.47	03 91 CA 3E F0 A3 D7
DelayEchoFilt	AD 44 65 6C 61 79 45 63 68 6F 46 69 6C 74
false	C2
{array of 5}	95
0, {arr 1}, 0.08	00 91 CA 3D A3 D7 0A
1, {arr 1}, 0.50	01 91 CA 3F 00 00 00
2, {arr 1}, 0.55	02 91 CA 3F 0C CC CD
3, {arr 1}, 0.37	03 91 CA 3E BD 70 A4
4, {arr 1}, 1.00	04 91 CA 3F 80 00 00
"bias.reverb	AB 62 69 61 73 2E 72 65 76 65 72 62
true	C3
{array of 8}	98
0, {arr 1}, 0.50	00 91 CA 3F 00 00 00
1, {arr 1}, 0.25	01 91 CA 3E 80 00 00
2, {arr 1}, 0.34	02 91 CA 3E AE 14 7B
3, {arr 1}, 0.49	03 91 CA 3E FA E1 48
4, {arr 1}, 0.34	04 91 CA 3E AE 14 7B
5, {arr 1}, 0.34	05 91 CA 3E AE 14 7B
6, {arr 1}, 0.30	06 91 CA 3E 99 99 9A
7, {arr 1}, 1.00	07 91 CA 3F 80 00 00
167	A7

The final byte is the checksum - this is calculated by adding all the data bytes in the msgpack data apart from the first two bytes, modulo 256.

Create the 'packed' 7 bit data with a chunk header.

The chunk is formatted like this:

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 8 bit xor is the xor of all the data bytes following the header so it does not include the header or the final 0xf7 trailer.

The data is the msgpack data but stored in 7 bit format. The first byte in each 8 byte slice contains the 8th bit of each of the following bytes. Bit 0 of that byte stores the 8th bit of the byte following and so on.

At the start of each data segment are three bytes which are inserted into the msgpack data – the total number of chunks, this chunk number (starting at 0) and the size of this chunk data.

The size of the data is the size of the actual msgpack data, so excludes these added three bytes and also all of the bytes storing the 8th bits.

This is usually 0x80 data bytes in a 'full' chunk being send. Because this is 7bit encoded this looks like 0x00 for the 'full' chunks.

```
F0 01 20 75 01 01
24 03 00 00 00 7F 59 24
00 62 61 36 66 34 32 31
00 38 2D 37 35 62 35 2D
00 34 30 32 64 2D 38 64
00 34 39 2D 33 39 39 32
00 38 36 64 62 38 35 61
02 61 29 41 43 34 30 20
10 54 6F 6E 65 23 30 2E
02 37 2B 4D 79 20 6F 77
40 6E 20 74 6F 6E 65 28
00 69 63 6F 6E 2E 70 6E
4A 67 4A 42 70 00 00 17
01 2E 62 69 61 73 2E 6E
00 6F 69 73 65 67 61 74
36 65 43 13 00 11 4A 3E
31 0F 5C 29 01 11 4A 3E
37 19 19 1A 02 11 4A 3F
09 00 00 00 2A 43 6F 6D
00 70 72 65 73 73 F7
```

```
F0 01 21 72 01 01
64 03 01 00 6F 72 43 12
36 00 11 4A 3E 42 0F 5C
76 01 11 4A 3E 28 75 43
01 27 42 6F 6F 73 74 65
36 72 43 11 00 11 4A 3F
0E 28 75 43 28 41 43 20
60 42 6F 6F 73 74 43 15
26 00 11 4A 3F 02 0F 5C
76 01 11 4A 3E 70 23 57
36 02 11 4A 3E 6B 05 1F
26 03 11 4A 3F 07 2E 14
```

```
66 04 11 4A 3E 19 19 1A
01 2C 43 68 6F 72 75 73
40 41 6E 61 6C 6F 67 42
0D 14 00 11 4A 3F 17 0A
2C 3D 01 11 4A 3E 33 33
4C 33 02 11 4A 3F 19 19
6D 1A 03 11 4A 3E 70 23
03 57 2D 44 65 6C F7
```

```
F0 01 22 26 01 01
00 03 02 76 61 79 45 63
40 68 6F 46 69 6C 74 42
6D 15 00 11 4A 3D 23 57
0C 0A 01 11 4A 3F 00 00
4C 00 02 11 4A 3F 0C 4C
2D 4D 03 11 4A 3E 3D 70
2D 24 04 11 4A 3F 00 00
02 00 2B 62 69 61 73 2E
40 72 65 76 65 72 62 43
0D 18 00 11 4A 3F 00 00
2C 00 01 11 4A 3E 00 00
2C 00 02 11 4A 3E 2E 14
6C 7B 03 11 4A 3E 7A 61
2C 48 04 11 4A 3E 2E 14
2C 7B 05 11 4A 3E 2E 14
6C 7B 06 11 4A 3E 19 19
2D 1A 07 11 4A 3F 00 00
02 00 27 F7
```

Create the block output

Prepend the block header of 16 bytes and adjust the length byte to be the length of the whole block.

```
01 FE 00 00 53 FE AD 00
00 00 00 00 00 00 00 00
F0 01 20 75 01 01
24 03 00 00 00 7F 59 24
00 62 61 36 66 34 32 31
00 38 2D 37 35 62 35 2D
00 34 30 32 64 2D 38 64
00 34 39 2D 33 39 39 32
00 38 36 64 62 38 35 61
02 61 29 41 43 34 30 20
10 54 6F 6E 65 23 30 2E
02 37 2B 4D 79 20 6F 77
40 6E 20 74 6F 6E 65 28
00 69 63 6F 6E 2E 70 6E
4A 67 4A 42 70 00 00 17
01 2E 62 69 61 73 2E 6E
00 6F 69 73 65 67 61 74
36 65 43 13 00 11 4A 3E
31 0F 5C 29 01 11 4A 3E
37 19 19 1A 02 11 4A 3F
09 00 00 00 2A 43 6F 6D
00 70 72 65 73 73 F7
```

```
01 FE 00 00 53 FE AD 00
00 00 00 00 00 00 00 00
F0 01 21 72 01 01
64 03 01 00 6F 72 43 12
36 00 11 4A 3E 42 0F 5C
76 01 11 4A 3E 28 75 43
01 27 42 6F 6F 73 74 65
36 72 43 11 00 11 4A 3F
0E 28 75 43 28 41 43 20
60 42 6F 6F 73 74 43 15
26 00 11 4A 3F 02 0F 5C
76 01 11 4A 3E 70 23 57
36 02 11 4A 3E 6B 05 1F
26 03 11 4A 3F 07 2E 14
66 04 11 4A 3E 19 19 1A
01 2C 43 68 6F 72 75 73
40 41 6E 61 6C 6F 67 42
0D 14 00 11 4A 3F 17 0A
2C 3D 01 11 4A 3E 33 33
4C 33 02 11 4A 3F 19 19
6D 1A 03 11 4A 3E 70 23
03 57 2D 44 65 6C F7
```

```
01 FE 00 00 53 FE A2 00
00 00 00 00 00 00 00 00
F0 01 22 26 01 01
00 03 02 76 61 79 45 63
40 68 6F 46 69 6C 74 42
6D 15 00 11 4A 3D 23 57
0C 0A 01 11 4A 3F 00 00
4C 00 02 11 4A 3F 0C 4C
2D 4D 03 11 4A 3E 3D 70
2D 24 04 11 4A 3F 00 00
02 00 2B 62 69 61 73 2E
40 72 65 76 65 72 62 43
0D 18 00 11 4A 3F 00 00
2C 00 01 11 4A 3E 00 00
2C 00 02 11 4A 3E 2E 14
6C 7B 03 11 4A 3E 7A 61
2C 48 04 11 4A 3E 2E 14
2C 7B 05 11 4A 3E 2E 14
6C 7B 06 11 4A 3E 19 19
2D 1A 07 11 4A 3F 00 00
02 00 27 F7
```

Preset format

Type	Length	Content	Example
Integer	1	0	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.	