Byte values -1 and 255.

Most often, an “unset” value for a parameter 0-127 is represented by -1 in the soundfont definitions (overridingRootKey, keynum, velocity) however for byOriginalPitch, and only byOriginalPitch, the “unset” value is specified as 255. This is a question of formal definition. Formally, byOriginalPitch is defined as a “BYTE” with a value in the range 0-255 and valid values from 0-127 rather than a “byte” which would have allowed “-1” to be used.

However, if you add 1 to 255 in a byte, the result is 0 because -1 and 255 are both correct representations of the byte 0xFF. It seemed to me that 0 saying was the closest valid value to -1 was less confusing than saying 0 was the closest valid value to 255, they are however equally true.

scaleTuning

There seem to be three main usages for scaleTuning.

1. Handling unpitched instruments
2. Simulating stretch tuning of pianos
3. Defining alternative even tempered scales

The first was my main interest, the second is essential for accurate rendering of pianos, and the third, although interesting, does not seem to be well supported in notation terms (MuseScore has quarter tone accidentals which represent only a very limited step in that direction) or in soundfont support (scaleTuning is a poor substitute for allowing keyNumber to be specified in cents).

There are two relevant passages in the specifications.

§8.1.2 “This parameter represents the degree to which MIDI key number influences pitch. A value of zero indicates that MIDI key number has no effect on pitch; a value of 100 represents the usual [even] tempered semitone scale.”

This certainly implies that 0 is a normal value and that it should be used to ensure that the “MIDI key number has no effect on pitch”: unpitched percussion is a prime example of this.

§9.1.1 “The pitch is described in terms of an initial pitch shift which is based on the sample’s sampling rate, the root key at which the sample should be unshifted on the keyboard, the coarse, fine, and correction tunings, the effective MIDI key number, and the keyboard scale factor”.

The initial pitch shift is the pitch shift before modulation and although this paragraph does not define the calculation, the mathematics are fairly basic.

If we set aside the coarse, fine, and correction tunings (coarseTune and fineTune in the instrument split and chCorrection in the sample) which have no incidence on the problem – they all default to 0 – and the sampling rates which do not have any incidence either, we are left with only three parameters “the root key at which the sample should be unshifted on the keyboard” (rootKey equal to either overridingRootKey or byOriginalKey), “the effective MIDI key number” (keyNumber) and the “keyboard scale factor” (scaleTuning).

Without scaleTuning, the pitch shift is expressed as

$$shift = 2^{\left(\frac{keyNumber - rootKey}{12}\right)}$$

If keyNumber is twelve greater than rootKey, the note frequency is doubled (moved up an octave or 12 semitones).

In general, introducing a linear scaling function requires the definition of the both the slope and the origin. In SF2 however, there is no way of specifying the origin, so for introducing keyNumber into the pitch shift function in accordance with §8.1.2 the standard function is

$$shift = 2^{\left(\frac{keyNumber - rootKey}{12} \times \frac{scaleTuning}{100}\right)}$$

This ensures that if scaleTuning is 100 then the result is “the usual [even] tempered semitone scale” and when it is zero, the result is “that MIDI key number has no effect on pitch) §8.1.2. It also ensures that a value of 1200 provides a shift of an octave per key (§8.1.3).

This standard function expands or compresses the scale about the root key. This is the normal behaviour of FluidSynth and every other synthesizer I have been able to test. This is not explicit in the SF2 specifications, but as there is no other parameter for changing this behaviour, it is implicit.

An alternative definition for scale tuning would be to use scaleTuning to scale keyNumber before it is used to calculate the shift. This would require a second root key to be specified for scale tuning. The scale would then be expanded or compressed about this scale root key.

For unpitched instruments (case 1 above) this is of no practical interest. For stretch tuning (case 2 above), this has an advantage over the standard function only for the unrealistic case of linear stretch. For a more realistic stretch using a piecewise linear approximation, it is slightly more complex than using the standard function.

It might, however, be useful as a patch for alternative even tempered scales (case 3 above) until fractional keyNumbers (also required for the non-functioning quarter note accidentals currently available in MuseScore) are implemented in the synthesizer.

$$shift = 2^{\left(\frac{scaleRoot + (keyNumber - scaleRoot) \times \frac{scaleTuning}{100} - rootKey}{12}\right)}$$

This alternative function is not supported in SF2 because there is no means of specifying a distinct scale root key and if scaleRoot is set to be equal to rootKey, then this alternative function becomes identical to the standard function.

Experimentation, however, has shown that this alternative function is used in MuseScore, with an arbitrary constant value of 60 for scaleRoot. THIS IS NOT “perfectly consistent with the Soundfont 2.1 specification” which has no mention of any parameter resembling scaleRoot and certainly no suggestion that 60 has any special meaning for scale tuning.