

SF-enhanced (SFe) 4

Technical Specification

Version 4.0b (9 February 2025)



Copyright © 2025 SFe Team and contributors

Copyright notice

All parts of this specification may be reproduced without additional written permission by the SFe Team, provided that you follow the license in section 2.3.

The SFe Team is in no way sponsored by or otherwise affiliated with Creative Technology Ltd. or E-mu Systems, Inc.

SoundFont® is a registered trademark of Creative Technology Ltd. The SoundFont® 2.04 file format is copyright © 1994–2002 E-mu Systems Inc.

To avoid confusion with the official SoundFont standard, the file format is referred throughout to as "SF" and/or "SFe". Future versions of SFe may not be compatible with legacy SF players.

That being said, "soundfont" is rapidly becoming a genericised word via the actions of the VGM (video game music) and lightsaber communities. If you are a member of the VGM, lightsaber, or another community, by not appropriating "SoundFont" for your unrelated formats, you can help keep the meaning of the trademark.

Any excerpts from SFSPEC24.PDF are copyrighted by Creative Technology Ltd, and are only used for reference. However, we are confident that the base file format is not copyrightable.

Disclaimers

This specification is subject to change without notice. Please obtain the latest version from the SFe Team GitHub page at <https://github.com/sfe-team-was-taken>.

This specification assumes familiarity of the SoundFont 2.04 file format (SFSPEC24.PDF), which can be found at <https://freepats.zenvoid.org/sf2/sfspec24.pdf>.

Table of contents

| | |
|--|-----------|
| Section 1 Introduction..... | 11 |
| 1.1 Preamble..... | 12 |
| 1.2 Changelog..... | 12 |
| 1.3 History of improvements to the SF format..... | 13 |
| 1.4 Scope and purpose..... | 14 |
| 1.5 Important differences from the legacy SF specification document..... | 14 |
| 1.6 Document organisation..... | 14 |
| Section 2 Credits and copyright..... | 17 |
| 2.1 Credits..... | 18 |
| 2.2 License..... | 18 |
| Section 3 Versioning and updates..... | 21 |
| 3.1 Specification Versioning..... | 22 |
| 3.2 Future improvements..... | 25 |
| 3.3 Long term support of SFe 4..... | 25 |
| Section 4 Terminology..... | 27 |
| 4.1 Data structure terminology..... | 28 |
| 4.2 Synthesis terminology..... | 30 |
| 4.3 Parameter terminology..... | 31 |
| Section 5 SFe file format structure..... | 33 |
| 5.1 File format extensions..... | 34 |
| 5.2 General RIFF-type format structures..... | 34 |
| 5.3 Chunk header types..... | 35 |
| 5.4 RIFF error checking features..... | 36 |
| 5.5 Structure of the SFe 4 file format..... | 36 |
| 5.5.1 SFe 4 file format structure outline..... | 36 |
| 5.5.2 ISFe-list information..... | 38 |
| 5.5.3 Changed and removed chunks..... | 38 |
| 5.5.4 String encoding..... | 39 |
| 5.6 INFO-list chunk..... | 39 |
| 5.6.1 ifil sub-chunk..... | 39 |
| 5.6.2 Versioning rules..... | 39 |
| 5.6.3 Specification versions to ifil values..... | 40 |
| 5.6.4 isng sub-chunk..... | 40 |
| 5.6.5 List of sound engines..... | 41 |
| 5.6.6 ICRD sub-chunk..... | 41 |
| 5.6.7 INAM, IENG, IPRD, ICOP, ICMT and ISFT sub-chunks..... | 42 |
| 5.6.8 irom and iver sub-chunks..... | 42 |
| 5.6.9 SFty sub-chunk..... | 42 |
| 5.6.10 SFvx sub-chunk..... | 43 |
| 5.6.11 flag sub-chunk..... | 45 |
| 5.7 sdta-list chunk..... | 46 |
| 5.7.1 smpl sub-chunk..... | 46 |
| 5.7.2 SFe Compression..... | 47 |

| | |
|--|-----------|
| 5.7.3 sm24 and sm32 sub-chunk..... | 49 |
| 5.7.4 Using 8-bit samples..... | 51 |
| 5.7.5 Looping rules..... | 52 |
| 5.8 pdta-list chunk..... | 53 |
| 5.8.1 phdr sub-chunk..... | 53 |
| 5.8.2 New Bank System..... | 53 |
| 5.8.3 inst sub-chunk..... | 57 |
| 5.8.4 shdr sub-chunk..... | 57 |
| 5.8.5 Other sub-chunks..... | 57 |
| Section 6 SFe enumerations and feature flags..... | 59 |
| 6.1 About SFe enumerations..... | 60 |
| 6.2 Feature flags..... | 60 |
| 6.2.1 Feature flag tree structure..... | 60 |
| 6.2.2 Branch 00 Foundational synthesis engine..... | 61 |
| 6.2.3 Branch 01 Modulators and NRPN..... | 66 |
| 6.2.4 Branch 02 Sample bitdepth support..... | 68 |
| 6.2.5 Branch 03 SFe Compression support..... | 69 |
| 6.2.6 Branch 04 Metadata upgrades..... | 69 |
| Section 7 Parameters and synthesis model..... | 71 |
| 7.1 About the synthesis model..... | 72 |
| 7.2 MIDI functions..... | 72 |
| 7.2.1 MIDI bank select..... | 72 |
| 7.2.2 Other MIDI functions..... | 72 |
| 7.3 Parameter units, generators, modulators and NRPNs..... | 72 |
| 7.4 On implementation accuracy..... | 72 |
| Section 8 SFe error handling..... | 75 |
| 8.1 Structurally Unsound errors..... | 76 |
| 8.2 Non-critical errors..... | 76 |
| 8.3 Duplicated preset locations within files..... | 77 |
| 8.4 Duplicated preset locations across files..... | 78 |
| 8.5 Undefined chunks..... | 78 |
| 8.6 Unknown Enumerators..... | 78 |
| 8.7 Maximum File Size Limit Exceeded..... | 79 |
| 8.8 MIDI Errors..... | 79 |
| 8.9 Illegal parameter values, out of range values, missing required items and illegal enumerators..... | 79 |
| Section 9 SiliconSFe..... | 81 |
| 9.1 SiliconSFe overview..... | 82 |
| 9.2 Header format..... | 82 |
| 9.2.1 About the header format..... | 82 |
| 9.2.2 romRiffHeader..... | 83 |
| 9.2.3 romByteSize..... | 83 |
| 9.2.4 romInterleaveIndex..... | 83 |
| 9.2.5 romRevision..... | 83 |
| 9.2.6 romVer..... | 83 |
| 9.2.7 bankChecksum..... | 83 |

| | |
|--|------------|
| 9.2.8 bankChecksum2sComplement..... | 83 |
| 9.2.9 bankSFeVersion..... | 84 |
| 9.2.10 bankProduct..... | 84 |
| 9.2.11 sampleCompType..... | 84 |
| 9.2.12 bankStyle..... | 84 |
| 9.2.13 bankCopyright..... | 84 |
| 9.2.14 romSFeBankStart..... | 84 |
| 9.2.15 romSineWaveStart..... | 84 |
| 9.2.16 sampleSineWave..... | 85 |
| 9.3 AWE ROM emulator..... | 85 |
| 9.3.1 Introducing the AWE ROM emulator..... | 85 |
| 9.3.2 ROM emulator sample specification..... | 86 |
| Section 10 Compatibility information..... | 97 |
| 10.1 Specification and structural compatibility..... | 98 |
| 10.1.1 Legacy SF2.04 specification compatibility..... | 98 |
| 10.1.2 INFO chunk and legacy compatibility..... | 98 |
| 10.1.3 phdr sub-chunk..... | 98 |
| 10.1.4 Player implementation quirks..... | 99 |
| 10.1.5 Generators and modulators..... | 100 |
| 10.1.6 Error handling..... | 101 |
| 10.2 Sample compatibility..... | 101 |
| 10.2.1 32-bit samples..... | 101 |
| 10.2.2 8-bit samples..... | 101 |
| 10.2.3 ROM samples..... | 101 |
| 10.2.4 Incompatible compression formats..... | 101 |
| Section 11 Program specification..... | 103 |
| 11.1 Program compatibility levels..... | 104 |
| 11.1.1 File format specifications..... | 105 |
| 11.1.2 Sample specifications..... | 106 |
| 11.1.3 Instrument specifications..... | 107 |
| 11.1.4 Preset specifications..... | 107 |
| 11.1.5 Player specifications..... | 108 |
| 11.2 Converting between legacy SF and SFe..... | 110 |
| 11.2.1 Conversion from legacy SF2.04 to SFe..... | 110 |
| 11.2.2 Conversion from SFe to legacy SF2.04..... | 110 |
| 11.2.3 Conversion from SFe to legacy SF2.01..... | 111 |
| 11.3 Converting between chunk header types..... | 111 |
| 11.3.1 Conversion of 32-bit static to 64-bit static..... | 111 |
| 11.3.2 Conversion of 64-bit static to 32-bit static..... | 111 |
| 11.3.3 Conversion between 32-bit static and RIFX..... | 112 |
| 11.4 File repair programs..... | 112 |
| 11.4.1 Repairing Structurally Unsound errors..... | 112 |
| 11.4.2 Repairing non-critical errors..... | 117 |
| 11.4.3 Manual repair..... | 122 |
| 11.4.4 Automatic repair..... | 122 |
| 11.5 Why these guidelines?..... | 124 |

| | |
|---|------------|
| 11.5.1 File Size Representation..... | 124 |
| 11.5.2 File Size Limit..... | 124 |
| 11.5.3 Sample Streaming..... | 124 |
| 11.5.4 Total File Size Limit and Multiple Files..... | 124 |
| 11.5.5 Legacy Support..... | 125 |
| 11.5.6 Header Support..... | 125 |
| 11.5.7 Sample Compression..... | 125 |
| 11.5.8 File Extension, Structure, Information and Metadata..... | 126 |
| 11.5.9 Sample Specifications..... | 126 |
| 11.5.10 Instrument Specifications..... | 126 |
| 11.5.11 Player Specifications..... | 127 |
| 11.6 How to test your program with SFSpecTest..... | 128 |
| 11.6.1 What does SFSpecTest do?..... | 128 |
| 11.6.2 Branch 00 Foundational synthesis engine..... | 128 |
| 11.6.3 Branch 01 Modulators and NRPN..... | 129 |
| 11.7 Courtesy actions..... | 129 |
| Section 12 Glossary..... | 131 |

Section 1

Introduction

1.1 Preamble

The SFe standard has been created to provide a successor to E-mu Systems®'s SoundFont® 2.04 standard.

- Large files (above 4 GiB) must use a 64-bit chunk header format.
- Programs that are designed to create existing SF2 files can easily be adapted to 64-bit with SFe.
- 64-bit headers will allow creators of sound banks to replace split banks with a large number of SF2 files, with large monolithic banks, where all samples and instruments can be used in every preset.
- SFe compatible players require a certain standard of features, which is in the program specification in section 11. This improves compatibility of SFe banks with players.

1.2 Changelog

| Revision | Date | Description |
|----------|-----------------|---|
| 4.0b | 9 February 2025 | Added base preset fallback Renamed "proprietary compression" to "incompatible compression" |
| 4.0a | 8 February 2025 | A few clarifications |
| 4.0 | 8 February 2025 | n/a |

For draft specification revision history, see `draft-revision-history.md` (available in the SFe specification package or on the GitHub repository).

Changes from the previous version of the specification are highlighted.

1.3 History of improvements to the SF format

In 2005, E-mu released SoundFont® 2.04 (specification date September 2002), the last version before abandoning the format. It included 24-bit support, and was designed for the Sound Blaster® X-Fi. Creative Technology Ltd is still around, but their current sound cards (such as the AE-9) do not support the SoundFont® format.

At an unknown time, Kenneth Rundt, the author of SynthFont and a series of products based on it, added some custom features, mostly ported over from the DLS format:

- Always play the sample to the end
- Velocity to volume envelope attack (from DLS)
- Velocity to modulation envelope attack (from DLS)
- Vibrato LFO to volume (from DLS)

Werner Schweer found a way to compress SoundFont® 2 files (with the lossy Vorbis format) around 2010.

- The resulting format was known as SF3.
- It is commonly used by open source programs, such as MuseScore and FluidSynth.
- This is the reason why this format is not called SF3.
- SFe incorporates the Werner SF3 specification as SFe Compression.

Another development was done by Cognitone, which created an open source program that can losslessly compress SoundFont® 2 files (with FLAC). This was done in 2017.

- This was unofficially called SF4.
- This is intended to be the true version 4, as Cognitone SF4 seems to not have been as widespread as SF3.
- SF4 has also been rejected by FluidSynth as too loosely defined.

Finally, stgiga found out that many programs don't mind RIFF64 (RF64) files.

- RIFF64 files have a much larger size limit than 32-bit RIFF files.
- It gives us a new horizon to experiment with longer, higher quality samples.
- There are also many other advantages of the RIFF64 format.

The SFe project started in 2020 as a proposal for a successor to legacy SF2.04 by Polyphone developer and SFe team member davy7125. Starting off as "SF3", it was renamed to avoid confusion with Werner SF3, firstly to "SF32 and SF64" but eventually to **SFe**.

1.4 Scope and purpose

This is the specification for the SFe 4.0 file format, based on the famous E-mu Systems® SoundFont® 2.04, and Werner SF3 standards, and is intended to be a source of information on SFe.

To create files that support SFe, you will need:

- The SFe technical specification (this document)
- E-mu's provided documents for legacy SF2.04 ([SFSPEC24.PDF](#))

All features from legacy SF2.04 will be retained.

1.5 Important differences from the legacy SF specification document

- This is not a standalone document; it refers to [SFSPEC24.PDF](#). All relevant information from the Werner SF3 specification is included.
- This document is unofficial and was not created by E-mu. If they want this to be removed, we will remove it, but we hope that this is because they have their own official update to the SF format ready.
- For copyright reasons, we cannot copy information from the original standard, except for what is required to interpret this document and what is allowed under fair use.
- The SFe specification includes many additional requirements over reading the file format.
- Document organisation has been significantly modified to accommodate the intricacies of the SFe format, and to make it easier to read.

1.6 Document organisation

The sections in this specification are different to the sections in [SFSPEC24.PDF](#), however they roughly correspond to some of these sections:

- Sections 1-3 of this specification to sections 0-1 of SFSPEC24.PDF
- Section 4 of this specification to section 2 of SFSPEC24.PDF
- Section 5 of this specification to sections 3-7 of SFSPEC24.PDF
- Section 6 of this specification to section 8 of SFSPEC24.PDF
- Section 7 of this specification to section 9 of SFSPEC24.PDF
- Section 8 of this specification to section 10 of SFSPEC24.PDF
- Section 9 of this specification to section 11 of SFSPEC24.PDF
- Section 12 of this specification to section 12 of SFSPEC24.PDF

Section 6 also contains feature flags along with enumerators, section 9 also contains information about the AWE ROM emulator that can be implemented, and sections 10 and 11 include information on compatibility concerns and guidance on writing SFe-compatible software respectively.

Significant differences from the previous version of the specification will be highlighted.

Section 2

Credits and copyright

2.1 Credits

SFe Team

Organisation

- GitHub: [SFe Team · GitHub](#)

Members

- sylvia-leaf (they/them) | Discord – @tanukilvia | Email – sylvialeaf6284@gmail.com
- stgiga (they/them) | Discord – @stgiga | Email – stgigamovement@yahoo.com
- spessasus
- davy7125

Want to join the SFe Team? Please contact sylvia-leaf using the above contact.

Special thanks

Thanks to these people or groups:

- derselbst (for format suggestions)
- mawe42 (for format suggestions)
- sagamusix (for feature suggestions)
- Werner Schweer (for creating Werner SF3, of which SFe Compression is based on)
- Falcosoft (for feature suggestions that will arrive in a future version of SFe)
- E-mu Systems (for creating the legacy SoundFont format, of which SFe is based on)

2.2 License

Copyright © 2020-2025 SFe Team and contributors

Permission is granted to use, distribute and modify this specification for any use, provided that:

- you attribute the SFe Team (do not remove copyright notices)
- you clearly mark any modifications that are made to the specification
- you do not claim that we're affiliated with E-mu or Creative Labs.

THIS SPECIFICATION IS PROVIDED "AS-IS" WITHOUT ANY WARRANTY WHATSOEVER, INCLUDING IMPLIED WARRANTY.

Section 3

Versioning and updates

3.1 Specification Versioning

Final specifications have version numbers in the format x.yL, where x and y are numbers and L is a letter:

- x is incremented when a change in the SFe format is made in a way that makes the resulting files incompatible with the previous version.
- y is incremented when there are new features added to the SFe format.
- SFe should not skip "y" versions.
- L is incremented when there are small changes made to the specification. The first version will not include L.
- Release candidates have very similar version numbers to final specifications, but include "rc" between y and L.
- An example of a final specification version would be 4.0.

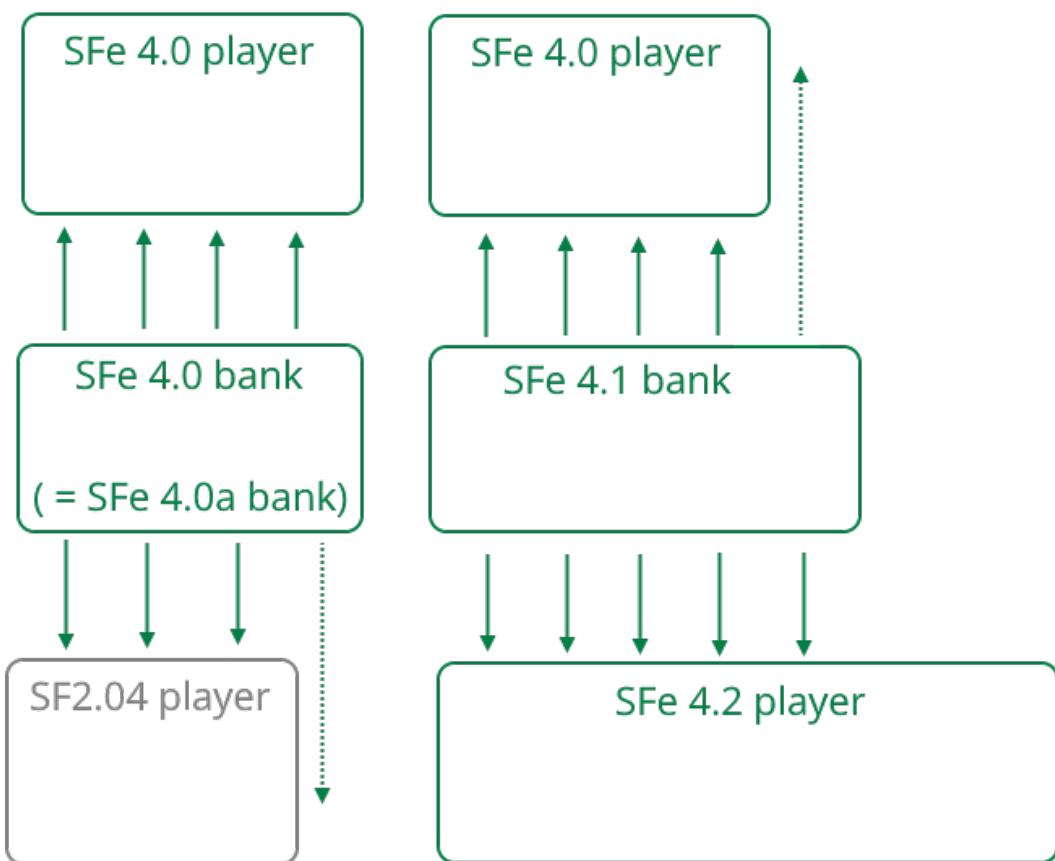


Figure 1: SFe 4.x versions are compatible with legacy players but with reduced sound quality. No changes are made to actual file structure in "L" versions. Later "y" versions are compatible with earlier players but with reduced sound quality.

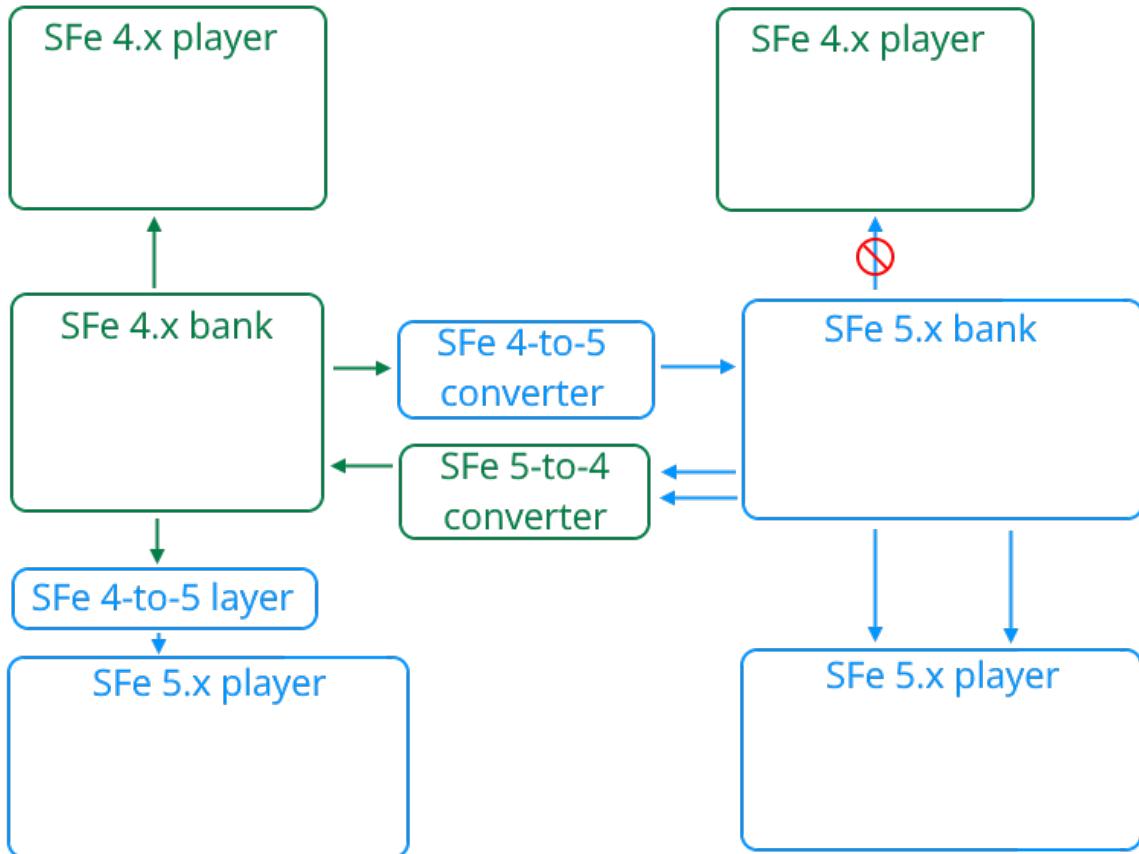


Figure 2: SFe "x" versions are not compatible with each other, but can be converted, resulting in reduced sound quality. Compatibility layers can be added to newer players to use older banks. It is also possible to upgrade banks to newer versions.

Draft specification milestones have version numbers in the format $x.y.zL$, where x , y and z are numbers and L is a letter. In this case, the versioning works similarly to a final specification, but with these changes:

- z is incremented when the draft undergoes a larger change, or large updates are made to the software.
- L is incremented when there are small changes, but only when pointed out by others. The first version will not include L .
- An example of a draft specification version would be 4.0.11a.

During the development of specifications, version numbers will be in the format $x.y.aaaabbccL$, where x , y , z , a , b and c are numbers, and L is a letter. The versioning is similar to final specifications and milestone drafts, but $aaaabbcc$ is the day in which the specification was updated, and L is incremented when updated.

Only final specifications are included in the changelog in section 1.2.

Draft specification

Always being updated

Make sure you have the latest version

x.y.aaaabbccL



Draft milestone

You can wait until the next draft milestone

We suggest getting the latest version

x.y.zL



Release candidate

Near-final versions of the specification

You can use these versions to create your software

x.y-rczL



Final version

The polished major releases

x.yL

Figure 3: The SFe development process.

3.2 Future improvements

SFe is designed for future improvements.

- These will be done in a more liberal way than the conservative manner of the SoundFont® 2 updates that E-mu has done.
- SFe is not affected by limitations created by the EMU8000 sound processor and Sound Blaster® cards, and therefore will have more improvements.
- Additionally, starting from SFe 5, SFe won't be affected by limitations created by legacy SF players.
- To avoid over-stress of developers of the SFe Team, as well as SFe instrument banks, features will be spread out across versions, and similar features will arrive at the same time.

Here are a few things that are planned for SFe:

- Polyphone 3 will be the first program that supports SFe. It will use it by default with legacy SF being an option. (Polyphone 2.5 seems to be planned as an LTS release for legacy SF.)
- Negotiations with more SF program developers such as FluidSynth and Bassmidi will start soon.
- For SFe 4.1, there will be an overhaul of the default modulators system, inspired by the [DMOD proposal by spessasus](#). Support for the related PNMM sub-chunk will also be included.
- A MIDI lyrics specification for MIDI players, along with Spessasus/Falcosoft RMIDI support, will become available in SFe 4.2.
- We will negotiate with the Synthfont author Kenneth Rundt about getting the Synthfont Custom Features added for SFe 4.2. Care has been taken to ensure that SFe parameter usage does not conflict with SFCF.

3.3 Long term support of SFe 4

SFe 4 is a "long-term support" version, and will get feature updates along with later versions such as SFe 5. While this is the case, some features are structurally incompatible with legacy SF2.04, and will not be available in SFe 5.

If an earlier major version of SFe is the main version used for a longer time than expected, then it can be declared as another LTS version. Such a LTS version will be declared in this section.

Section 4

Terminology

4.1 Data structure terminology

The data structure terminology used in SFe 4.0 is broadly the same as legacy SF2.04, with these additions:

- Branch – A subdivision of a tree structure containing either sub-branches or leaves that include values.
- BW64 – Broadcast Wave 64, used in the RF64 Header.
- Cognitone SF4 – An incompatible modification to Werner SF3 to allow support for FLAC audio compression. Because it is considered an incompatible compression format, usage is not allowed in SFe.
- FLAC – A lossless audio compression format commonly used in open-source software. Supported by Werner SF3, but not commonly used for that purpose.
- Leaf – A value found in a tree structure at the end of a branch.
- Lossless compression – Said of a compression format that retains all of its data when compressed. In terms of audio, there is no loss in quality in losslessly compressed audio.
- Lossy compression – Said of a compression format that does not retain all of its data when compressed. In terms of audio, there is a loss in quality in lossily compressed audio.
- OGG – See "Vorbis".
- Opus – A lossy audio compression format, slightly newer than OGG but with less wide adoption.
- Quirk – Any player-specific function that is automatically enabled and modifies the behaviour of any numeric parameters used by legacy SF2.0x, including preset locations, parameters, units, modulators or NRPNs.
- Quirks mode – A mode in an SFe-compatible player that enables the implementation quirks.
- RF64 – See "RIFF64".
- RIFF64 – A 64-bit RIFF-type format. Contrast to 32-bit versions of the RIFF format. Therefore, the maximum file size is above 4 gigabytes in size.
- RIFF-type format – Formats similar to RIFF (Resource Interchange File Format), see "RIFF" in SF3SPEC24.PDF for more information.

- SFe – A family of enhancements to the SoundFont® 2.04 formats, unofficially created after E-mu/Creative abandoned the original format. May not be structurally compatible with legacy SF2.04.
- SFe 4 – This new specification, based on SoundFont® 2.04 and Werner SF3, with a set of new features making it more realistic. Not to be confused with the incompatible Cognitone SF4 file format.
- SFe-compatible – Indicates files, data, synthesisers, hardware or software that conform to the SFe specification.
- SFe Compression – The compression system based on Werner SF3 that SFe programs should be compliant with.
- Static RIFF – Any RIFF-type format with a fixed chunk size field width, including RIFF or RIFF64. See "RIFF-type format", "RIFF" and "RIFF64".
- Tree structure – A structure consisting of branches and leaves.
- Vorbis – A lossy audio compression format commonly used in open-source software. The basic compression format that most Werner SF3 and SFe-compatible software should be expected to implement.
- Werner SF3 – A small upgrade to SoundFont® 2.04 created by Werner Schweer to allow an open source compression solution for SoundFont® programs. Standardised as SFe Compression.

And these changes:

- RIFF – The 32-bit static RIFF format used by SoundFont® 2.04 and the unmodified version of the format described by Microsoft in 1991.

4.2 Synthesis terminology

The synth terminology used in SFe 4.0 is broadly the same as legacy SF2.04, with these additions:

- AWE64 – The successor to the famous AWE32, adding features such as waveguide synthesis. Used the EMU8000 synthesizer chip, like the preceding AWE32. Available in "Value" or "Gold" versions.
- DAHDSR – Stands for Delay, attack, hold, decay, sustain, release. The six-step envelope system used in SF and SFe.
- EMU10K1 – The successor to the EMU8000, designed by E-mu® for the Creative Labs SB Live!.
- EMU10K2 – An update to the EMU10K1, designed by E-mu® for the Creative Labs SB Audigy.
- EMU20K1 – The successor to the EMU10K2, designed by E-mu® for the Creative Labs SB X-Fi.
- EMU20K2 – An update to the EMU20K1, please refer [here](#) for information on SB X-Fi cards that include it.
- Hold – The portion of the DAHDSR envelope after the attack portion, but before the decay portion starts.
- Legacy sound card – A Sound Blaster® (or other sound card) that uses a hardware MIDI synthesiser capable of using banks in the SoundFont® format.
- ROM samples – Obsolete feature used in legacy sound cards, most modern SF2 files do not use this feature.
- Sharp – Said of a tone that is higher in pitch than another reference tone.
- SB – Abbreviation of "Sound Blaster®". For example, "SB X-Fi".
- Sound Blaster® Live! – The successor to the AWE64, which improved the synthesizer chip to the EMU10K1, supporting modulators.
- Sound Blaster® Audigy – The successor to the SB Live!, containing the EMU10K2 chip.
- Sound Blaster® X-Fi – The successor to the SB Audigy, containing the EMU20K1 or EMU20K2 chip. Supports 24-Bit SoundFont® 2 files (2.04).
- Synth – Abbreviation of "Synthesiser," see "Synthesiser" in SFSPEC24.PDF for more information.

These changes:

- Articulation – Modulation of available parameters and usage of extra samples to produce expressive musical notes.
- Case-insensitive – Indicates that a UTF-8 character or string treats alphabetic characters of upper or lower case as identical.
- Case-sensitive – Indicates that a UTF-8 character or string treats alphabetic characters of upper or lower case as distinct.
- Downloadable – legacy SF2.0x, Werner SF3 or SFe file obtained from the internet. (Old meaning referred to the obsolete ROM system)
- MIDI Bank – Groups of up to 128 presets, which can be selected by the two MIDI "Bank Select" control changes (CC00 and CC32).

And these removals:

- Predator (refers to an old legacy SF2.0x editor made by E-mu)

4.3 Parameter terminology

The parameter terminology used in SFe 4.0 is broadly the same as legacy SF2.04, with these additions:

- Amplification – An increase in volume or amplitude of a signal.
- Flat – Said of a tone that is lower in pitch than another reference tone.

Section 5

SFe file format structure

5.1 File format extensions

The file format extension to use for SFe files is generally `.sf4`:

- `.sf2` is avoided because SFe files are *not* SoundFonts, but simply files that use formatting that is very similar to legacy SF2.04.
- `.sf3` is avoided because some Werner SF3 bank players may not support SFe features.

Despite `.sf4` also being used by cognitone-formatted banks, these banks never existed due to a [fatal bug](#) in cognitone's `sf2convert` program.

The presence of a legacy SF file extension such as `.sf2` or `.sf3` does not necessarily denote a legacy SF bank! SFe-compatible programs are expected to parse the `ifil` value and `ISFe-list` sub-chunk to properly load the bank, regardless of the extension.

The file type should be referred to as `SFe bank` and should *not* be referred to as `SoundFont` or anything containing `SoundFont`. `SFSPEC24.PDF` states that files with additional chunks don't conform to the legacy SF2.04 standard.

SFe currently does not use a MIME type.

5.2 General RIFF-type format structures

RIFF-type formats are the file format used in legacy SF2.04, Werner SF3 and SFe standards.

There are a few different RIFF-type format structures:

- RIFF is the basic version with 32-bit chunk headers, and is used in legacy SF2.04 and Werner SF3.
- RIFF64 (also called RF64) is mostly compatible with RIFF, but uses 64-bit chunk headers.
- RIFX is a big-endian version of 32-bit RIFF, while RIFF/RIFF64 are little-endian formats.

RIFF-type formats are created in building blocks known as "chunks."

Chunks are defined using this structure:

- `ckID`: type of data in chunk, equal to a unique 4-character code (FourCC), listed above.
- `ckSize`: size of chunk (RIFF, RIFX), equal to 4,294,967,295 (RIFF64)
- `ds64`: size of chunk (RIFF64 only)
- `ckDATA[ckSize]`: the data inside the chunks, including pad bytes.

Chunks can be further divided into "sub-chunks."

Orders of chunks in all SFe banks are strictly defined, as in legacy SF2.04, and should be kept to, except for TSC mode.

5.3 Chunk header types

In SFe, there are different chunk header types that are used in the format. These correspond to different RIFF-type formats. Currently, there are two defined chunk header types:

- 32-bit static
 - This is the same as legacy SF
 - This corresponds to RIFF.
 - The FourCC used is RIFF.
- 64-bit static
 - A ds64 chunk is added
 - The FourCC used is RF64.
 - To prevent loading by incompatible players, the sfbk fourcc is replaced with sfen (**SF-enhanced**)
 - This corresponds to RIFF64.

Future versions of SFe may define different chunk header types.

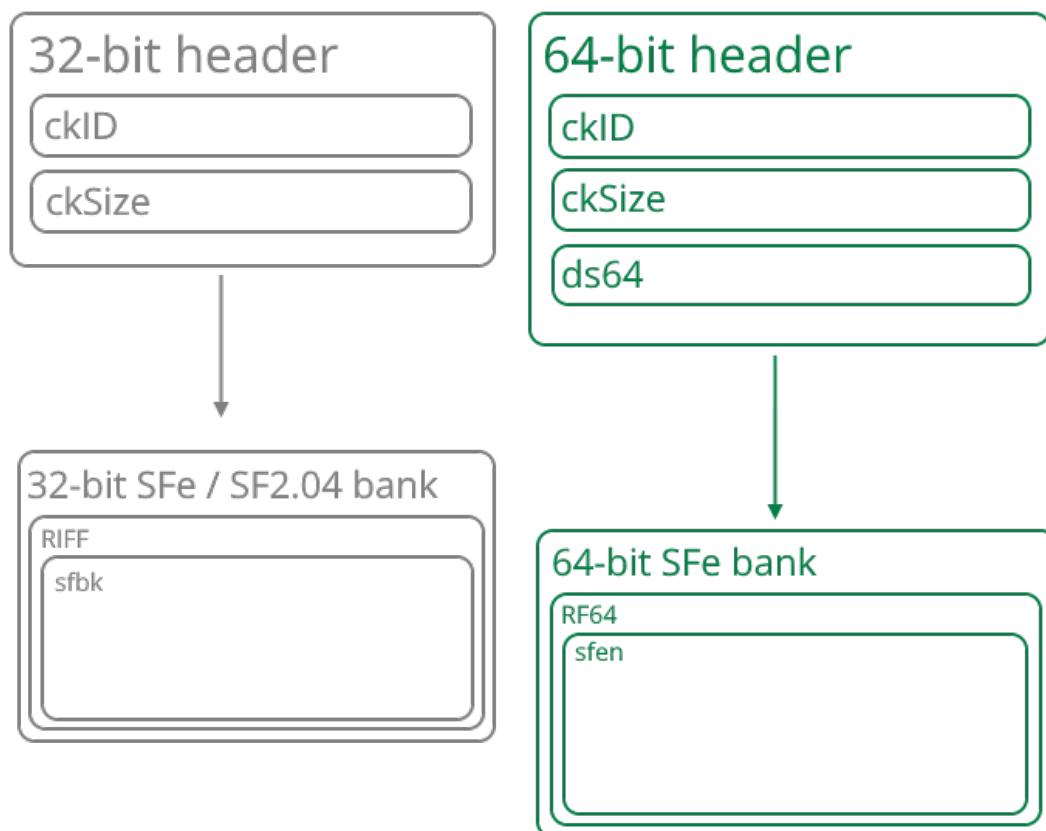


Figure 4: 32-bit static versus 64-bit static headers.

5.4 RIFF error checking features

RIFF-type formats have error checking features about:

- The size of the file
- The length of the chunks
- The length of the sub-chunks

Using this information, it is possible to check for damage to an SF(e) file.

5.5 Structure of the SFe 4 file format

5.5.1 SFe 4 file format structure outline

An SFe 4 file consists of:

- RIFF chunk (main chunk) - this changes depending on the chunk header type to be used.
 - sfbk ascii string – use sfen with 64-bit chunk headers
 - LIST
 - INFO ascii string
 - Sub-chunks inside INFO-list in legacy SF2.04 – ifil, isng, etc.
 - LIST
 - ISFe ascii string
 - Chunks listed in section 5.5.2
 - LIST
 - sdt_a ascii string
 - Sub-chunks inside sdt_a-list in legacy SF2.04 – smpl, sm24
 - sm32 chunk (BYTE array)
 - LIST
 - pdt_a ascii string
 - Sub-chunks inside pdt_a-list in legacy SF2.04

Only SFe-specific chunks are listed for brevity. In this section, assume that any non-listed chunk is identical to legacy SF2.04.

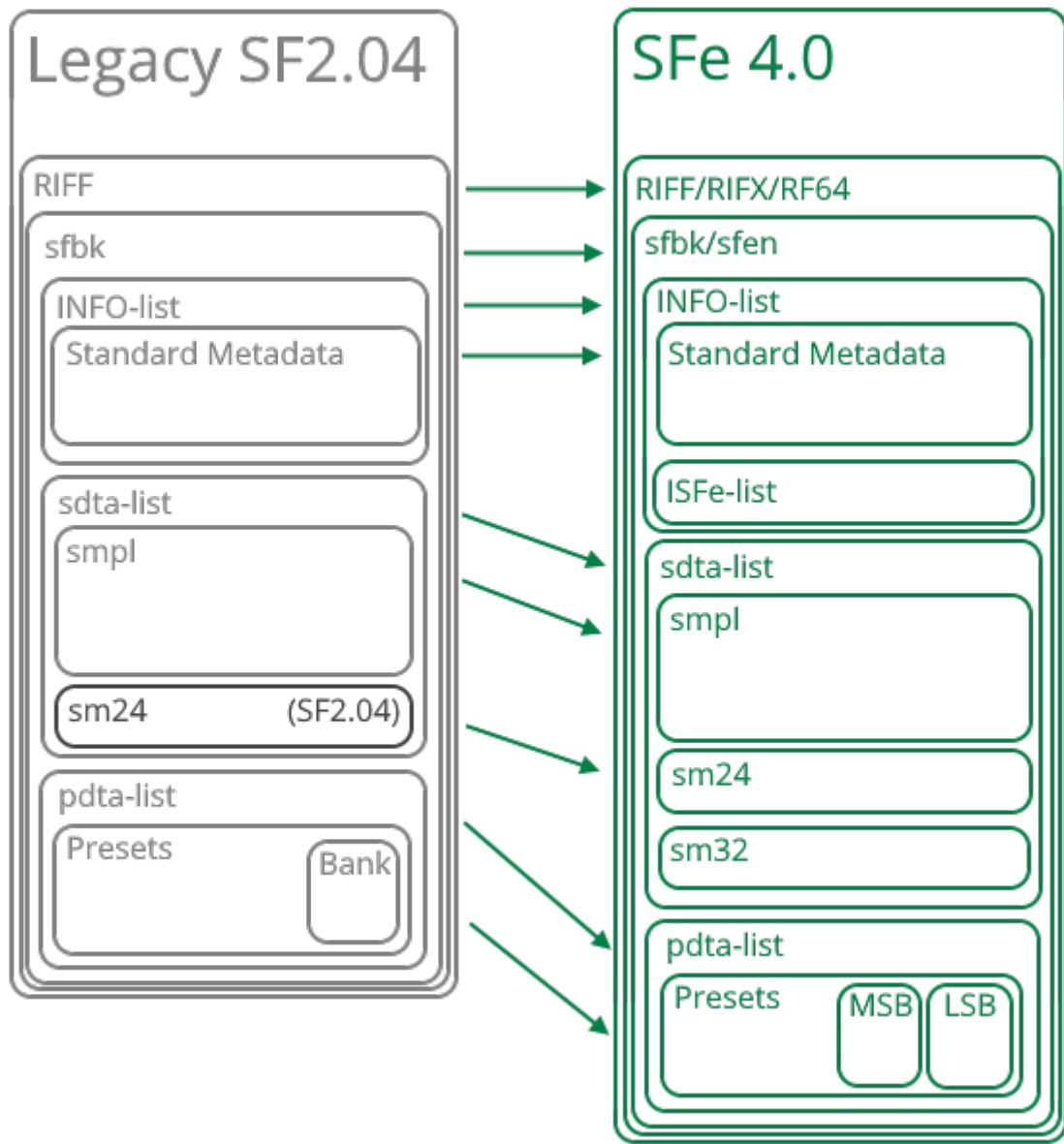


Figure 5: Legacy SF2.04 vs SFe 4.0 structures.

5.5.2 ISFe-list information

The ISFe-list sub-chunk includes many different sub-chunks to show information about SFe-specific features. Generally, we use the ISFe-list sub-chunk to make it clearer that this kind of information is SFe-specific.

Due to possible compatibility constraints, the ISFe-list sub-chunk is found inside the INFO-list sub-chunk, rather than as a fourth RIFF chunk. At least a few legacy soundcards (notably the SB X-Fi) do not error out on the inclusion of a fourth separate chunk. Officially, according to SFSPEC24.PDF, additional sub-chunks mean that an SFe file is not conformant to the legacy SF2.04 standard.

The ISFe-list sub-chunk currently contains these sub-chunks as of version 4.0:

- SFty chunk (UTF-8 string)
- SFvx chunk (46 bytes)
 - wSFeSpecMajorVersion (WORD)
 - wSFeSpecMinorVersion (WORD)
 - achSFeSpecType [20] (CHAR)
 - wSFeDraftMilestone (WORD)
 - achSFeFullVersion [20] (CHAR)
- flag chunk (multiple of 6 bytes)
 - byBranch (BYTE)
 - byLeaf (BYTE)
 - dwFlags (DWORD)

5.5.3 Changed and removed chunks

In the pdta-list sub-chunk, the wBank chunk has been replaced by byBankMSB and byBankLSB. These are functionally the same, but expressed in a different way to make the specification more readable.

5.5.4 String encoding

For most string fields, the encoding to use is now UTF-8 instead of ASCII. Mojibake may result on legacy SF players when using characters unsupported by ASCII. Because some characters use multiple bytes in UTF-8, you may not be able to use as many multi-byte characters compared to single-byte characters.

This applies to the `isng`, `INAM`, `irom`, `ICRD`, `IENG`, `IPRD`, `ICOP`, `ICMT`, `ISFT` chunks from legacy SF2.04, as well as the `achPresetName` (`PHDR`), `achInstrumentName` (`INST`) and `achSampleName` (`SHDR`) fields.

5.6 INFO-list chunk

5.6.1 ifil sub-chunk

The value of the `ofil` sub-chunk is equal to 2.1024 or 3.1024 depending on whether SFe Compression is used.

The size must be exactly four bytes. Reject files with an `ofil` sub-chunk that isn't four bytes as Structurally Unsound.

If the `ofil` sub-chunk is missing, either:

- Assume version 4.0.
- Reject the file as Structurally Unsound.

5.6.2 Versioning rules

In SFe 4.0, new versioning rules are used to replace the old ones.

The value of `wMajor` increases every time a change is made to the format that makes it incompatible with existing players.

- There will be at least 6 months between the first draft milestone of a new `wMajor` version and the release of the final specification.
- Older `wMajor` versions must be supported, either directly or via translation to the latest version.
- We strive to minimize the number of these updates whenever possible in favor of updates that are backward compatible.

The value of `wMinor` increases every time a change is made to the `SFvx` sub-chunk while retaining backwards compatibility.

- These updates generally have only one or two draft milestones before the final specification releases.
- Feature updates in these versions are smaller.

The specification type used is found in the ISFe-list sub-chunk.

5.6.3 Specification versions to ifil values

| wSFeSpecMajorVersion | wSFeSpecMinorVersion | ofil (wMajor) | ofil (wMinor) |
|----------------------|----------------------|---------------|---------------|
| 4 | 0 | 2 or 3 | 1024 |

5.6.4 isng sub-chunk

A new default isng sub-chunk value is used in SFe: SFe 4.

- SFe 4.0 players should recognize this and remove the default velocity related filter used in legacy SF2.04.
- In the case of a missing isng chunk, files with an ifil sub-chunk with wMajor = 2 or 3 and wMinor >= 1024, or wMajor >= 4, assume an isng sub-chunk value of SFe 4. Don't assume EMU8000.

Additionally, UTF-8 is now used instead of ASCII, and the length limit is removed.

- The isng sub-chunk contains a UTF-8 string of any length.
- Example of value: SFe 4 (with appropriate zero bytes).

Reject anything not terminated with a zero byte, and assume the value SFe 4. Do NOT assume EMU8000 by default.

5.6.5 List of sound engines

Creative/E-mu

| Sound engine name | isng value | Creative/E-mu SF version | Bit depth | Sound cards |
|-------------------|------------|--------------------------|-----------|--------------|
| EMU8000 | EMU8000 | 1.0, 1.5, 2.00 | 16 bit | AWE32, AWE64 |
| EMU10K1 | E-mu 10K1 | 2.01 | 16 bit | SB Live! |
| EMU10K2 | E-mu 10K2 | 2.01 | 16 bit | SB Audigy |
| EMU20K1, EMU20K2 | X-Fi | 2.04 | 24 bit | SB X-Fi |

SFe

| Sound engine name | isng value | SFe version | Bit depth |
|-------------------|----------------|-------------|------------------------|
| SFe 4 | SFe 4 | 4.0 | 16 bit, 24 bit, 32 bit |
| SFe 4 quirks mode | SFe 4 (quirks) | 4.0 | 16 bit, 24 bit |

5.6.6 ICRD sub-chunk

To ease the creation of library management systems that are compatible with multiple languages, the naming convention for the ICRD sub-chunk has been changed.

The value of ICRD must now be compliant with the ISO-8601 standard. There are two valid formats:

- Date only: for example 2025-02-08
- Date and time: for example 2025-02-08T02:28:00Z

Library management systems should be able to read the value of the ICRD sub-chunk and show the date (and time if applicable) in the correct language in a field that can be sorted.

If the value of the ICRD sub-chunk is missing or not in any of the above two valid formats, the program may either:

- attempt to parse the value (if the chunk is present)
- ignore the value (if present) and show an "unknown" error on the date (and time) field
- overwrite the value with the current date (and time) if the program is an editor

The program must NOT reject a file with a missing or invalid ICRD sub-chunk as Structurally Unsound.

5.6.7 INAM, IENG, IPRD, ICOP, ICMT and ISFT sub-chunks

These sub-chunks are mostly the same as in legacy SF2.04, but UTF-8 is now used instead of ASCII, and the length limit is removed.

Reject anything not terminated with a zero byte. Do NOT reject the file as Structurally Unsound.

5.6.8 irom and iver sub-chunks

Read the legacy SF2.04 specification for info on how to use ROM samples.

The ROM emulator should be implemented in SFe programs.

5.6.9 SFty sub-chunk

The SFty sub-chunk is required and contains a case-sensitive UTF-8 string with even length identifying the type of format used in SFe. Its value is used by SFe-compatible players to assist in loading banks by telling the program what variant of SFe to load a bank as.

The defined values of the SFty chunk are:

- the 12 bytes representing SFe-static as 10 UTF-8 characters followed by two zero bytes.
- the 20 bytes representing SFe-static with TSC as 19 UTF-8 characters followed by one zero byte.
- the 20 bytes representing SFe-static (8-bit) as 18 UTF-8 characters followed by two zero bytes.
- the 28 bytes representing SFe-static (8-bit) with TSC as 27 UTF-8 characters followed by one zero byte.

The field should not be longer than 28 bytes in SFe 4.0.

If the SFty sub-chunk is missing or its contents are an undefined value or in an invalid format, other properties of the structure should be used to determine the variant of SFe that is in use. Do not assume SFe-static; only use such a value when it is evident beyond a reasonable doubt that the file used is in the SFe-static format.

5.6.10 SFvx sub-chunk

The SFvx sub-chunk is required and contains extended SFe version attributes. It is always 46 bytes in length, containing data in the structure below:

```
struct SFeExtendedVersion
{
    WORD wSFeSpecMajorVersion;
    WORD wSFeSpecMinorVersion;
    CHAR achSFeSpecType[20];
    WORD wSFeDraftMilestone;
    CHAR achSFeFullVersion[20];
};
```

The WORD values wSFeSpecMajorVersion and wSFeSpecMinorVersion contain the SFe specification version, and are used to differentiate between different SFe versions as the value of ifil only changes when the format of the `SFvx` sub-chunk does so.

The case-sensitive UTF-8 character field achSFeSpecType contains a specification type in UTF-8. For the purposes of this specification, the defined values are:

- Final for final specifications.
- Release Candidate for release candidate specifications.
- Milestone for draft specification milestones.
- Dev for rolling draft specifications.

Assume Final if contents are unknown.

The WORD value wSFeDraftMilestone contains the draft specification milestone or release candidate number that a bank was created to. This varies depending on the value of achSFeSpecType.

The case-sensitive UTF-8 character field achSFeFullVersion contains the full version string of the specification used, for example 4.0a.

If the SFvx sub-chunk is missing or of an incorrect size, assume these values:

- wSFeSpecMajorVersion and wSFeSpecMinorVersion correspond to the highest version declared in the flag sub-chunk
 - If there is no valid flag sub-chunk, then assume the highest SFe version supported by the program.
- achSFeSpecType=Final
- wSFeDraftMilestone=0
- achSFeFullVersion corresponds to the other assumed values

The file may optionally be rejected as Structurally Unsound.

5.6.11 flag sub-chunk

The flag sub-chunk is required and contains the feature flags used by a bank. It is always a multiple of 6 bytes in length, and contains at least 2 records (1 feature flag and a record at the end) according to the structure:

```
struct SFeFeatureFlag
{
    BYTE byBranch;
    BYTE byLeaf;
    DWORD dwFlags;
};
```

The BYTE value `byBranch` represents the branch of the feature. Branches correspond to types of features.

The BYTE value `byLeaf` represents the leaf of the feature. Leaves correspond to specific features.

The DWORD value `dwFlags` represents the feature flags themselves, which represent different parts of the feature. Depending on the `byLeaf` value, it can be a number, a series of bytes, etc.

A tree value is a combination of a branch value and a leaf value, and is conventionally written in the format [branch] : [leaf] with hexadecimal values, for example "feature flag 03:01" refers to the feature flag with branch number 3 and leaf number 1 (SFe Compression sample compression formats). While the `flag` sub-chunk uses a tree structure, it should be noted that no branch includes sub-branches; the branches only include leaves.

Branch numbers between 240 (F0) and 255 (FF) are private-use branches that will not be defined in the SFe specification itself, and are free to be used by programs.

An exhaustive list of feature flags and their corresponding tree values can be found in section 6.2.

The final record should never be accessed in normal usage, but its value of `byBranch` and `byLeaf` have strict values depending on the specification version. Any records after the terminal record or with a higher tree value combination (except for the defined private-use area) should be ignored.

If the flag sub-chunk is missing or an incorrect size, then an effort should be made to recover the data. If data is not recoverable, then it can be rebuilt from the properties of the data in the rest of the bank. Do not reject the file as Structurally Unsound.

5.7 sdata-list chunk

5.7.1 smpl sub-chunk

This sub-chunk will now be present in most SFe files, as there is likely to be no ROM where samples can be read from. This does not include AWE ROM emulation. It works in an almost identical manner to legacy SF2.04, with these important differences:

- This contains one or more samples of audio in linearly coded 16-bit, signed words. These words are little-endian if the header is RIFF or RF64, and big-endian if the header is RIFX.
- No more leeway of 46 zero-valued samples is required after each sample.
- Before saving, SFe editors should insert this leeway. Otherwise, they might give a warning telling the user that loop and interpolation quality may be affected.
- If ROM samples are detected in SFe files, attempt to load them, even if this sub-chunk is missing.
- If this sub-chunk is missing, and no ROM samples are found, show a suitable error message.

5.7.2 SFe Compression

To implement compression in your SFe bank, please use the SFe Compression specification, listed in this section (5.7.2).

What is SFe Compression?

SFe Compression is the compression encoding system used by SFe, based on the earlier [Werner SF3](#) system widely used by the open source community.

By standardising on Werner SF3 in the form of SFe Compression, we will hopefully ensure that everyone uses the same compression formats. Due to this, we will only make small changes to SFe Compression which correspond to updates to the Werner SF3 specification by other SF player programs. To achieve this, all SFe players should implement SFe Compression.

File identification for SFe Compression

The `wMajor` value in the `ifil` sub-chunk is set to 3 instead of 2. The value of the `SFvx` sub-chunk remains unchanged. Therefore, SFe players should not use the `ifil` value to determine the SFe version, but rather the `SFvx` sub-chunk.

sfSampleType in shdr sub-chunk

Bit 4 of the `sfSampleType` field indicates a sample that has received some kind of compression. While most Werner SF3 compatible programs compress the samples using the Vorbis format, it cannot be assumed. SFe players must determine the encoding that is used for each sample.

For a sample to be valid:

- the type of compression and/or encoding must be recognised and supported by the program.
- the compression and/or encoding format must be valid.
- the compressed sample must only have one channel.

A sample is not valid if any of these conditions are not true. If a sample is not valid, then all instruments and presets that use the sample should be rejected.

Interpretation of sample data index fields in shdr sub-chunk

If bit 4 of the sfSampleType field is set, then the interpretation of the four sample data index fields changes:

- dwStart points to the first byte of the compressed byte stream relative to the beginning of the `smp1` sub-chunk.
- dwEnd points to the last byte of the compressed byte stream, instead of the first zero-valued sample data point after the sample data in legacy SF2.04.
- dwStartLoop and dwEndLoop specify loop points relative to the start of the individual uncompressed sample data, in sample data points.

If bit 4 of the sfSampleType field is clear, then the sample data index fields should be interpreted as in legacy SF2.04.

Using both compressed and uncompressed samples in the same file

You can use both compressed and uncompressed samples with SFe Compression. Simply place uncompressed PCM samples at the beginning of the `smp1` sub-chunk before the compressed sample byte stream. Because each sample is compressed individually, the resulting byte streams of all encoded samples are written to the `smp1` sub-chunk. The `smp1` sub-chunk may also contain uncompressed little-endian PCM samples.

For compressed byte streams, it is not necessary to add forty-six zero-valued sample data points after each sample. The length of the `smp1` sub-chunk is not required to be a multiple of two for compressed banks, and its surrounding LIST chunk is also not padded to a multiple of two as a consequence.

Unsupported features

Compressed samples are always 16-bit samples. They do not make use of the `sm24` (or `sm32`) sub-chunk. If an `sm24` sub-chunk is present, its respective byte counterparts to the compressed byte stream stored in the `smp1` sub-chunk remain unused. Since all uncompressed PCM samples are stored before compressed samples in the `smp1` sub-chunk, the size of the `sm24` sub-chunk is minimised. The `sm24` size constraint defined in legacy SF2.04 therefore no longer applies in compressed banks.

Sample links are not used in banks compressed with SFe Compression. The value of `wSampleLink` should be read and written as zero.

Incompatible compression formats

The only supported compression system for SFe is the Werner SF3-compatible SFe Compression. Incompatible SF compression formats (`.sfark`, `.sfpack`, `.sf2pack`, `.sfogg`, `.sfq`, `.sf4`) must not be used. Because Cognitone SF4-formatted banks are not valid Werner SF3 banks, they are also incompatible with SFe Compression.

5.7.3 sm24 and sm32 sub-chunk

These sub-chunks are optional.

- The sm32 sub-chunk contains the least significant byte, and the sm24 sub-chunk contains the next least significant byte after sm32.
- Each sub-chunk is exactly half the size of the smpl sub-chunk for uncompressed banks. This may not apply when SFe Compression is in use.
- For every two bytes in the smpl sub-chunk, there is one byte in these sub-chunks (if all samples are compressed).
- These sub-chunks can only be used with uncompressed samples. This limitation may be removed in future versions of SFe.

If these sub-chunks are present, they are combined with the other sub-chunks to create a sample with higher bitdepth.

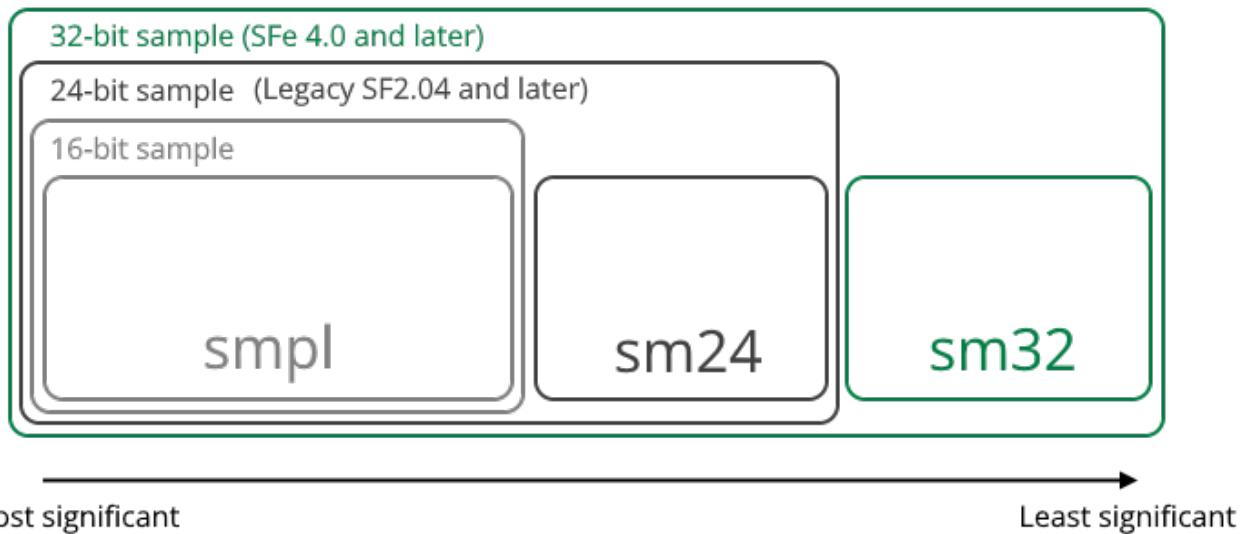


Figure 6: Available sample types in standard sample mode.

- If the `ifil` version is below 2.04 (signifying legacy SF2.01 or earlier), both `sm24` and `sm32` are ignored.
- If the `ifil` version is exactly 2.04 (signifying legacy SF2.04), only `sm32` is ignored. The `sm24` sub-chunk is still used.
- For uncompressed banks, these sub-chunks are not exactly half the size of the `smp1` sub-chunk (or otherwise invalid), the data should be ignored.
- If only the `sm32` sub-chunk is invalid, the `sm24` sub-chunk should still be loaded.
- However, if only the `sm24` sub-chunk is invalid, both sub-chunks should be ignored.
- If the `sm24` sub-chunk is ignored, the synthesizer should only attempt to render the first 16 bits of the samples contained within the `smp1` sub-chunk.
- If only the `sm32` sub-chunk is ignored, the synthesizer should attempt to render both the `smp1` and `sm24` sub-chunks, resulting in a 24-bit sample.
- We recommend only using `sm32` with 64-bit chunk headers. Using `sm32` with a 32-bit chunk header is syntactically valid, but it is not practical due to file size limitations.

5.7.4 Using 8-bit samples

If the `smpl` sub-chunk is missing, but the `sm24` or `sm32` sub-chunks are present, then the `sm24` or `sm32` sub-chunk is considered "orphaned".

In this case, the value of the `SFty` sub-chunk inside the `ISFe-list` sub-chunk should be checked. If it is equal to `SFe-static` (8-bit) or `SFe-static` (8-bit) with `TSC`, then 8-bit sample mode is to be activated.

- The sample depth is eight bits.
- The length of the `sm24` sub-chunk should be rounded up to the nearest byte, as if the sub-chunk was being used with a `smpl` sub-chunk.
- This mode can only be used if all samples are 8-bit. You cannot mix 8-bit and higher sample depths.
- SFe 4 does not support non-standard sample bit depths (6-bit, 12-bit, 18-bit, etc.)

If there is an orphaned `sm24` or `sm32` sub-chunk, and the `SFty` sub-chunk is missing or is not equal to a value that corresponds to 8-bit samples being present, then the file should be rejected as Structurally Unsound.

If there is both an orphaned `sm24` and an orphaned `sm32` sub-chunk, the `sm24` sub-chunk is the most significant byte, and 16-bit sample playback is assumed. Editing software should give a warning if there is a `sm24` and a `sm32` sub-chunk but no `smpl` sub-chunk, and should convert it to a 2.01-compliant 16-bit format. This behaviour should be followed if supported, regardless of whether 8-bit sample playback support is actually supported.

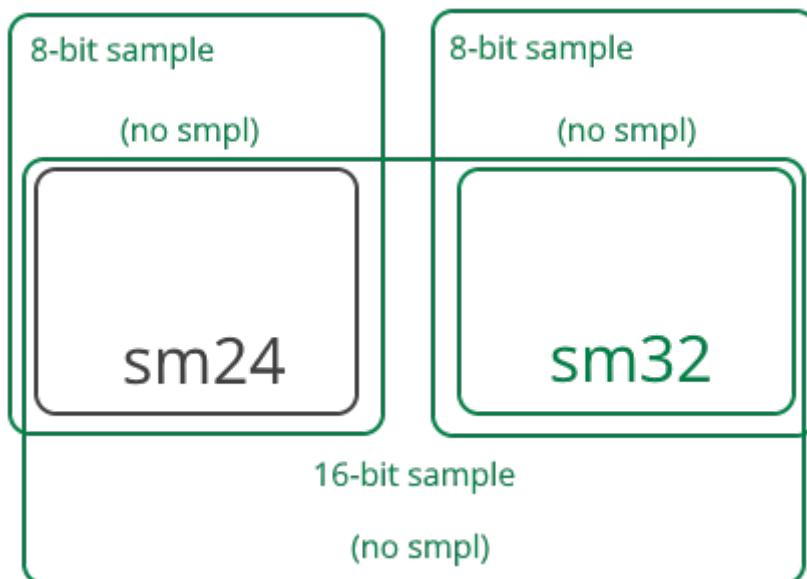


Figure 7: Available sample types in 8-bit sample mode.

5.7.5 Looping rules

- No more leeway of eight samples is required.
- Before saving, SFe editors might give a warning about this leeway telling the user that loop and interpolation quality may be affected.

5.8 pdta-list chunk

5.8.1 phdr sub-chunk

Its size is a multiple of 38 bytes, and its structure is the same as in legacy SF2.04.

The last sfPresetHeader entry shouldn't need to be accessed, apart from the uses described in SFSPEC24.PDF. The phdr sub-chunk is required; files without a phdr sub-chunk are Structurally Unsound.

5.8.2 New Bank System

In SFe 4.0, the bank system has been completely overhauled. Please read this section carefully to ensure that you correctly implement bank selects in your program.

Using MIDI Control Change #32 (Bank Select LSB)

In legacy SF2.04, the wBank field stores the bank that the preset can be found in. Due to a forward-thinking decision by E-mu, it is a WORD (16-bit) instead of a BYTE (8-bit). This means that it could theoretically store values for both bank select instructions found in MIDI 1.0.

Bank select LSB support is added by using the unused 8 bits of wBank according to the figure below. Bits 2-8 of *both* byBankMSB and byBankLSB are now used to set a bank change.

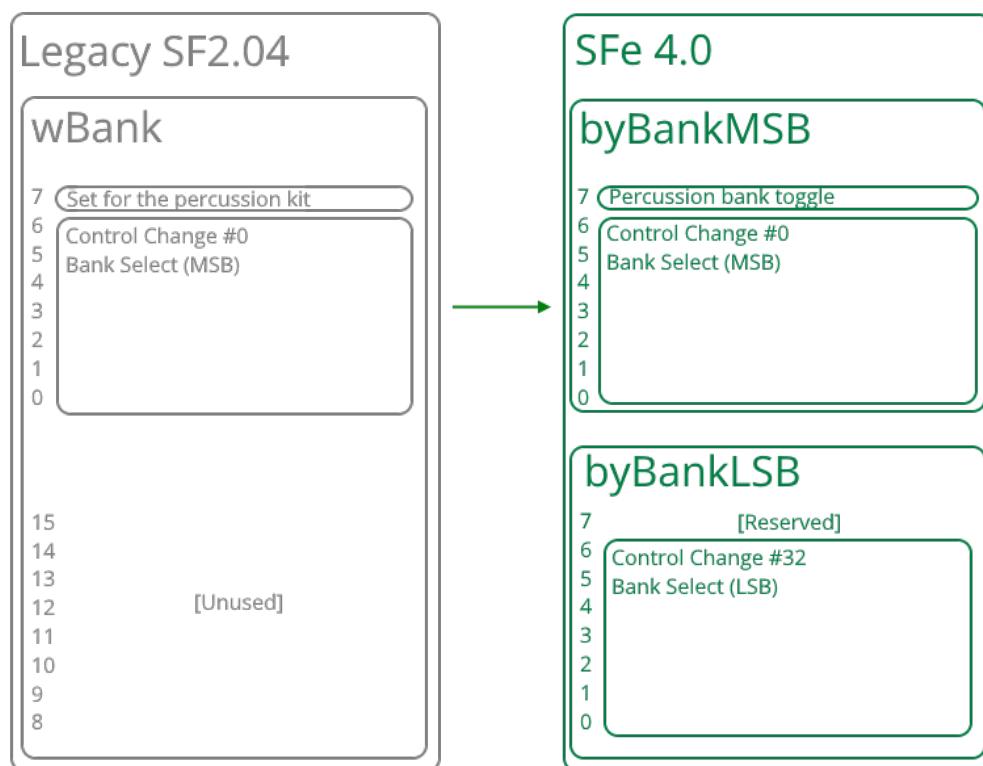


Figure 8: How the bank select logic differs from legacy SF2.04.

Introducing byBankMSB and byBankLSB

In the above figure, wBank has been replaced with byBankMSB and byBankLSB.

This splits the one WORD in legacy SF2.04 into two BYTE values, one for each bank. byBankMSB goes before byBankLSB due to RIFF being a little-endian format. For RIFX, byBankLSB is first.

Using more than one percussion bank

Legacy SF2.04 allows bank developers to define one bank of percussion kits for use in channel 10 that can be switched between using MIDI Program Change instructions by using the wBank number 128. In other words, if bit 7 is set, bits 0-6 must be clear - you cannot use bank select instructions with channel 10.

SFe 4.0 now allows users to set bit 7 with any value for bits 0-6. The result is that there are 128 percussion banks available when using byBankMSB, as shown by the figure below.

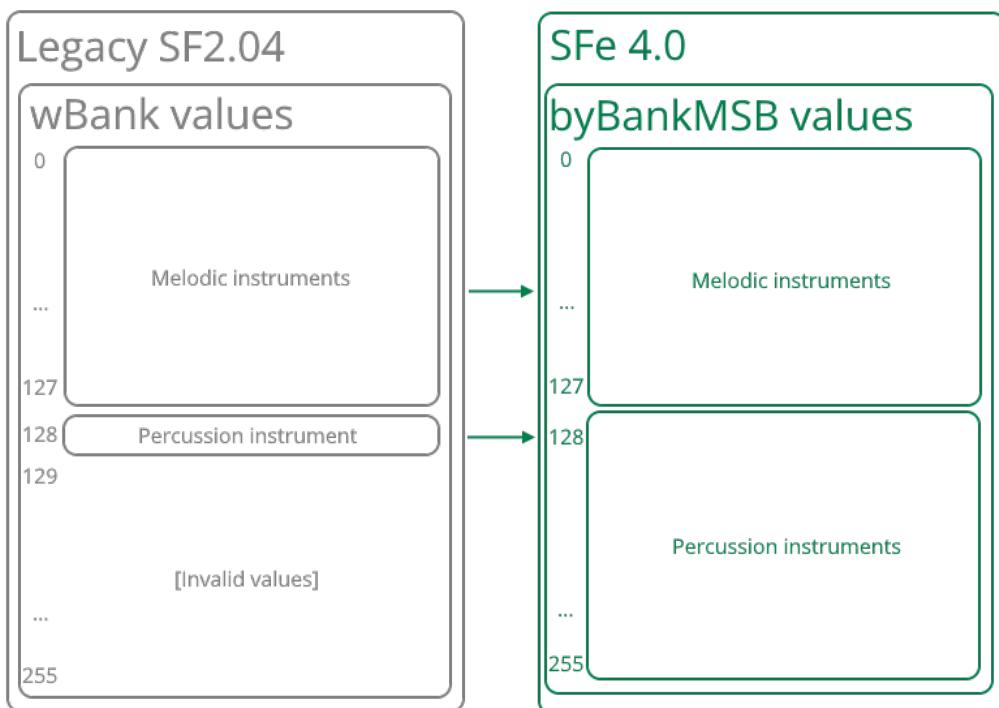


Figure 9: How the percussion bank listing differs from legacy SF2.04. When byte 7 is set for byBankMSB, byBankLSB may also be used. Therefore, a total of 16384 (128×128) banks of percussion kits may be used.

Flowchart for correct handling of bank select instructions

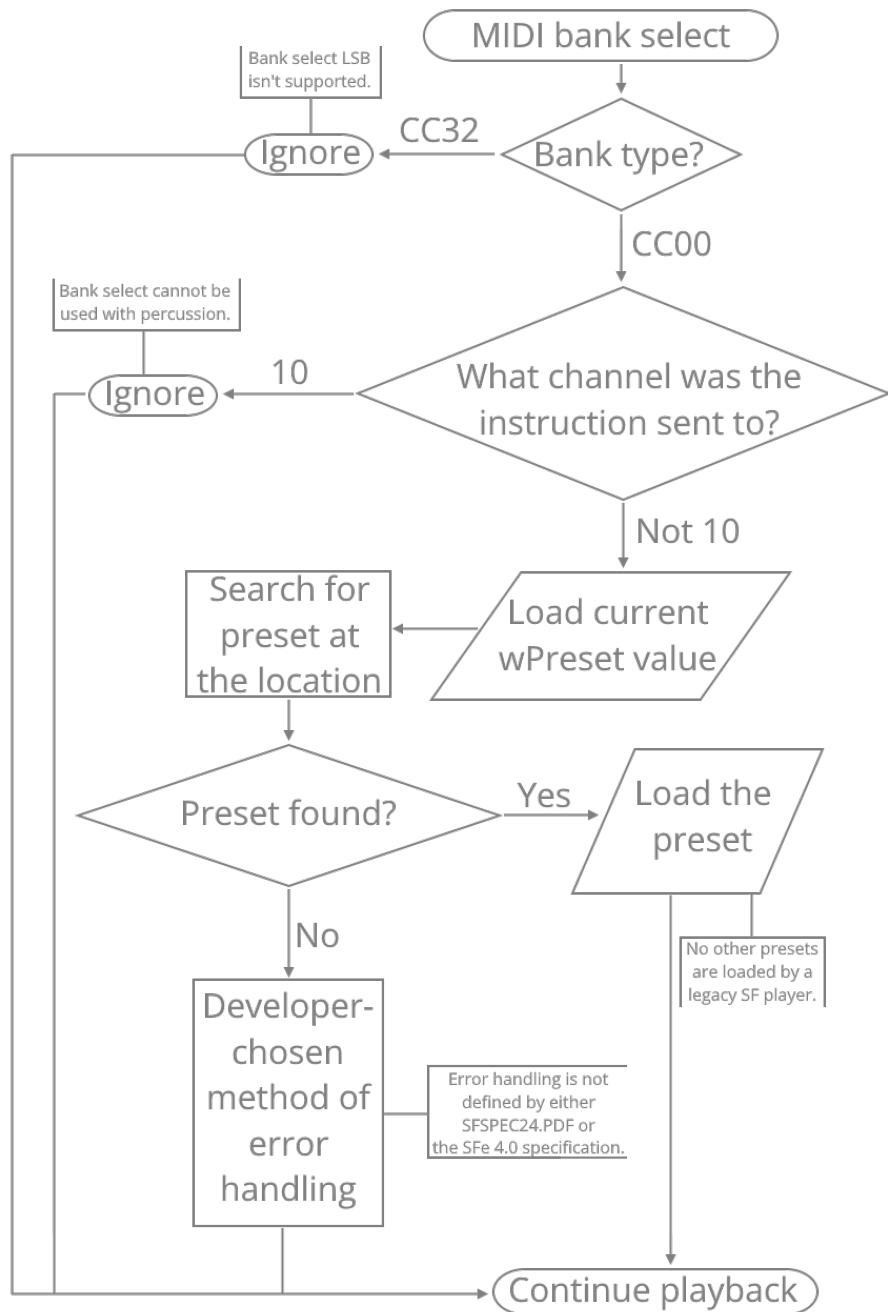


Figure 10: The flowchart for bank select instructions in legacy SF2.04.

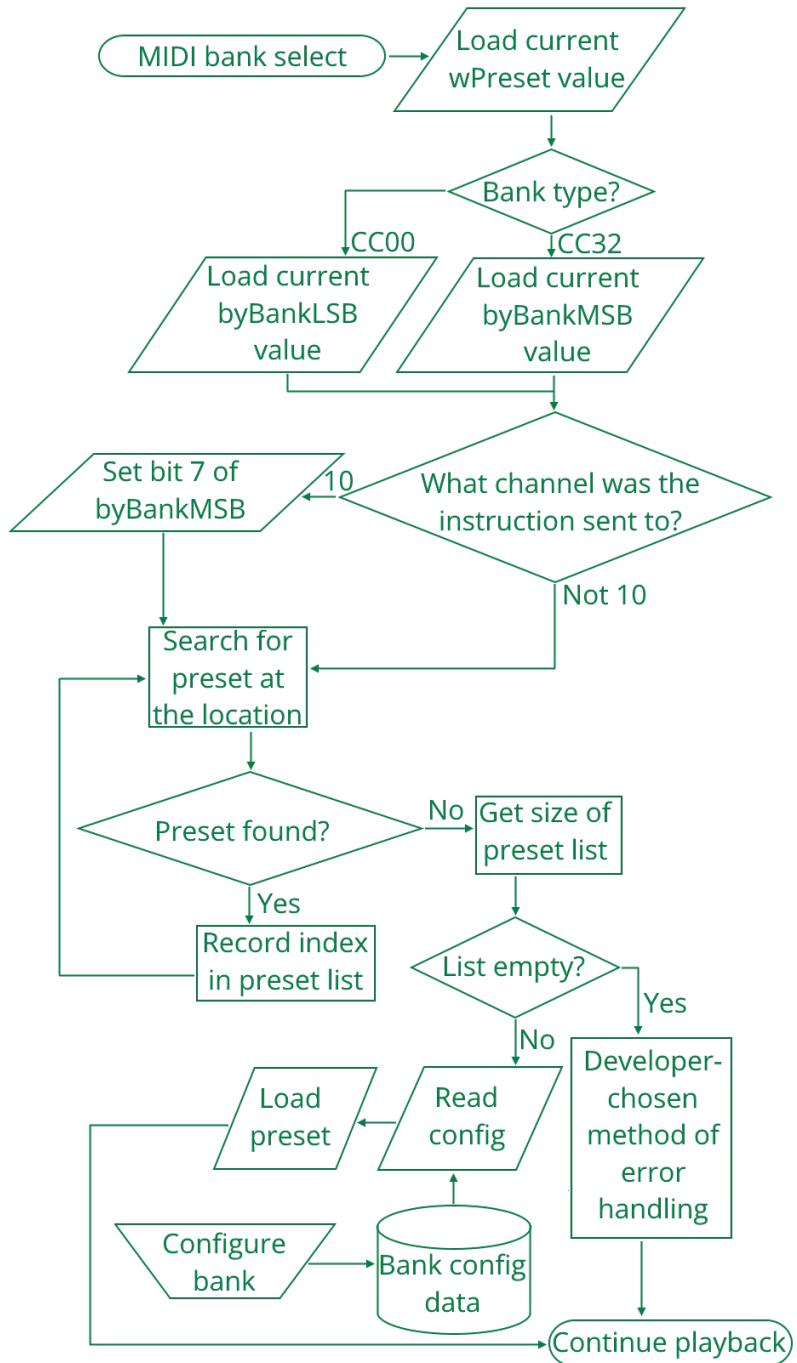


Figure 11: The flowchart for bank select instructions in SFe 4.0.

Notice that not only are extra steps added for bank select LSB and percussion bank select handling, but extra configuration information used by the player is added to determine the correct preset to use.

5.8.3 inst sub-chunk

Its size is a multiple of 22 bytes, and its structure is the same as in legacy SF2.04.

The `inst` sub-chunk is required; files without an `inst` sub-chunk are Structurally Unsound.

5.8.4 shdr sub-chunk

Its size is a multiple of 46 bytes, and its structure is the same as in legacy SF2.04.

The `shdr` sub-chunk is required; files without a `shdr` sub-chunk are Structurally Unsound.

Sample Rate Limit Changes

- In SFe, sample rates (`dwSampleRate`) are stored as a 32-bit integer. This is the same behavior as seen in the legacy SF2.04 format. This results in a theoretical maximum sample rate of 4,294,967,295 Hz.
- In the legacy SF2.04 specification, E-mu suggested that sample rates of below 400 Hz or above 50,000 Hz should be avoided as some legacy hardware platforms may not be able to reproduce these sounds. This is not a limitation of the specification, but rather a limitation of legacy sound cards.
- Despite this, Creative did not use 16-bit integers for sample rate in legacy SF2.04. It is thus safe to use sample rates in excess of 50,000 Hz. If a sample rate of below 400 Hz or above 50,000 Hz is encountered, no attempt should be made to change the sample rate.
- A zero sample rate should be reset.

sfSampleType and SFe Compression

Bit 4 of `sfSampleType` is reserved for SFe Compression usage.

- Read section 6.2 for more information on SFe Compression!

5.8.5 Other sub-chunks

The `pbag`, `pmod`, `pgen`, `ibag`, `imod` and `igen` sub-chunks work in the same way as legacy SF2.04. Read SFSPEC24.PDF for more information.

If any of the sub-chunks listed above is missing or invalid, the SFe bank is Structurally Unsound.

Section 6

SFe enumerations and feature flags

6.1 About SFe enumerations

SFe 4.0 enumerations are identical to legacy SF2.04, but more enumerations may be defined in the future.

Once the enumeration model is changed significantly, we will list all changed modulator and generator enums with a clear explanation of what they do. Equations for timecents will also be included!

6.2 Feature flags

6.2.1 Feature flag tree structure

The feature flags system is split like this:

- Branches: Roughly corresponds to each version (but not all listed features are part of the version). Maximum of 256. Number may increase with later wMajor versions.
- Leaves: Corresponds to each feature. Maximum of 256. These may change with later wMajor versions. Contains 32-bit data declaring how much of the feature is implemented.
- Flags: Each of the 32 bits that comprise a leaf, declaring support for specific features.

Feature flags listed as "reserved" must not be used for private use. Branches 240 (F0) to 255 (FF) are provided for such use.

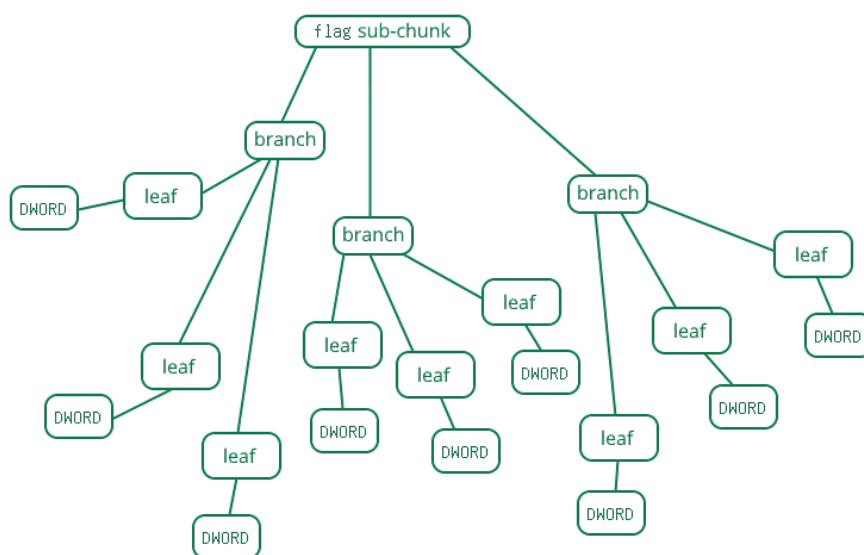


Figure 12: The tree structure of the feature flags system.

6.2.2 Branch 00 Foundational synthesis engine

00:00 Tuning

- Bit 1: Coarse tuning
- Bit 2: Fine tuning
- Bit 3: Root key
- Bit 4: Scale tuning

00:01 Looping

- Bit 1: Loop during release
- Bit 2: Non-loop during release

00:02 Filter Types

- Bit 1: Sound Blaster compatible low pass (12dB filter)

00:03 Filter Parameters

- Bits 1-16: Maximum supported filter frequency/cutoff
- Bits 17-24: Maximum supported filter resonance
- Bits 25-32: Reserved

00:04 Amplification and attenuation

- Bit 1: Attenuation supported in preset (0.4x)
- Bit 2: Attenuation supported in instrument (0.4x)
- Bit 3: Amplification supported in preset (0.4x)
- Bit 4: Reserved
- Bit 5: Reserved
- Bit 6: Reserved
- Bit 7: Reserved

00:05 Effects blocks

- Bit 1: Instrument-level reverb
- Bit 2: CC91 reverb
- Bit 3: Combined reverb
- Bit 4: Adjustable reverb
- Bit 9: Instrument-level chorus
- Bit 10: CC93 chorus
- Bit 11: Combined chorus
- Bit 12: Adjustable chorus
- Bit 17: Pan supported

00:06 Low Frequency Oscillators

- Bit 1: Vibrato supported
- Bit 2: Pitch Modulation
- Bit 3: Filter Modulation
- Bit 4: Amplitude Modulation

00:07 Envelopes

- Bit 1: Volume delay
- Bit 2: Volume attack
- Bit 3: Volume hold
- Bit 4: Volume decay
- Bit 5: Volume sustain
- Bit 6: Volume release
- Bit 7: Key to volume hold
- Bit 8: Key to volume decay
- Bit 9: Modulation delay
- Bit 10: Modulation attack
- Bit 11: Modulation hold
- Bit 12: Modulation decay
- Bit 13: Modulation sustain
- Bit 14: Modulation release
- Bit 15: Key to modulation hold
- Bit 16: Key to modulation decay
- Bit 17: Modulation of volume
- Bit 18: Modulation of pitch
- Bit 19: Modulation of filter

00:08 MIDI Control Changes

- Bit 1: 00 Bank Select MSB
- Bit 2: 00 Bank Select MSB for percussion
- Bit 3: 06 Data Entry MSB
- Bit 4: 32 Bank Select LSB (Multiple banks)
- Bit 5: 32 Bank Select LSB (Preset name)
- Bit 6: 32 Bank Select LSB (byBankLSB support)
- Bit 7: 32 Bank Select LSB for percussion
- Bit 8: 38 Data Entry LSB
- Bit 9: 64 Sustain
- Bit 10: 66 Soft
- Bit 11: 67 Sostenuto
- Bit 12: 98 NRPN LSB
- Bit 13: 99 NRPN MSB
- Bit 14: 100 RPN LSB
- Bit 15: 101 RPN MSB
- Bit 16: 120 All sound off
- Bit 17: 121 Reset all controllers
- Bit 18: 123 All notes off
- Bit 19: Reserved

00:09 Generators

- Bit 1: Index gen support
- Bit 2: Range gen support
- Bit 3: Substitution gen support
- Bit 4: Sample gen support
- Bit 5: Value gen support
- Bit 6: PGEN support
- Bit 7: IGEN support

00:0a Zones

- Bit 1: Key range
- Bit 2: Velocity range
- Bit 3: Exclusive class
- Bit 4: Fixed key
- Bit 5: Fixed velocity
- Bit 6: Sample offset
- Bit 7: Loop offset

00:0b Reserved

- Bit 1: Reserved
- Bit 2: Reserved
- Bit 3: Reserved
- Bit 4: Reserved

6.2.3 Branch 01 Modulators and NRPN

01:00 Modulators

- Bit 1: Primary source
- Bit 2: Secondary source
- Bit 3: Transform support
- Bit 4: Linear curves
- Bit 5: Concave curves
- Bit 6: Convex curves
- Bit 7: Switch curves
- Bit 8: Positive unipolar
- Bit 9: Negative unipolar
- Bit 10: Positive bipolar
- Bit 11: Negative bipolar
- Bit 12: Modulator chaining
- Bit 13: PMOD support
- Bit 14: IMOD support

01:01 Modulation controllers

- Bit 1: Note-on velocity
- Bit 2: Note-on key number
- Bit 3: Poly pressure
- Bit 4: Channel pressure
- Bit 5: Pitch wheel
- Bit 6: Pitch wheel sensitivity

01:02 Modulation parameters 1

- Bits 1-32: MIDI CC000-031

01:03 Modulation parameters 2

- Bits 1-32: MIDI CC064-095

01:04 Modulation parameters 3

- Bits 1-32: MIDI CC096-127

01:05 NRPN

- Bit 1: NRPN select MSB=120
- Bit 2: NRPN select LSB: 1-2 digits
- Bit 3: NRPN select LSB: 3 digits
- Bit 4: NRPN select LSB: 4 digits
- Bit 5: NRPN select LSB: 5 digits

01:06 Default modulators

- Bit 1: MIDI note on to initial attenuation
- Bit 2: MIDI note on to filter cutoff
- Bit 3: MIDI channel pressure to vibrato LFO pitch depth
- Bit 4: MIDI CC1 to vibrato LFO pitch depth
- Bit 5: MIDI CC7 to initial attenuation
- Bit 6: MIDI CC10 to pan position
- Bit 7: MIDI CC11 to initial attenuation
- Bit 8: MIDI CC91 to reverb send
- Bit 9: MIDI CC93 to chorus send
- Bit 10: MIDI pitch wheel to initial pitch, controlled by pitch wheel sensitivity
- Bit 17: MIDI note on to filter cutoff (SF2.00)
- Bit 18: MIDI note on to filter cutoff (SF2.01)
- Bit 19: MIDI note on to filter cutoff (SF2.04)
- Bit 20: Reserved
- Bit 21: Reserved
- Bit 24 off, bit 25 off: Reserved
- Bit 24 on, bit 25 off: Reserved
- Bit 24 on, bit 25 on: Reserved

01:07 Reserved

- Bit 1 off, bit 2 off: Reserved
- Bit 1 on, bit 2 off: Reserved
- Bit 1 on, bit 2 on: Reserved
- Bit 3: Reserved
- Bit 4: Reserved
- Bit 5: Reserved
- Bit 6: Reserved

01:08 Reserved

- Bit 1: Reserved
- Bit 2: Reserved
- Bit 3: Reserved
- Bit 4: Reserved
- Bit 5: Reserved

6.2.4 Branch 02 Sample bitdepth support

02:00 24-bit support

- Bit 1 off, bit 2 off: No support
- Bit 1 on, bit 2 off: Read support only
- Bit 1 on, bit 2 on: Playback support
- Bit 3: 8-bit support

02:01 32-bit support

- Bit 1 off, bit 2 off: No support
- Bit 1 on, bit 2 off: Read support only
- Bit 1 on, bit 2 on: Playback support
- Bit 2: Support for combining two 8-bit chunks into a 16-bit sample

6.2.5 Branch 03 SFe Compression support

03:00 Compression flag

- 0: sfSampleType bit 4 unsupported
- 1: sfSampleType bit 4 supported

03:01 Sample compression formats

- Bit 1: OGG
- Bit 2: Opus
- Bit 3: FLAC
- Bit 4: MP3

6.2.6 Branch 04 Metadata upgrades

04:00 Metadata improvements

- Bit 1: UTF-8 in INFO
- Bit 2: UTF-8 in pdta

04:01 Reserved

- Bit 1: Reserved
- Bit 2: Reserved
- Bit 3: Reserved

04:02 User-defined sample ROMs

- Bit 1: Support for user-defined sample ROMs

04:03 ROM emulator

- Bit 1: 1MB ROM emulator support

04:04 Reserved

- Bit 1: Reserved

Section 7

Parameters and synthesis model

7.1 About the synthesis model

Currently, the synthesis model of SFe 4.0 is identical to that of legacy SF 2.04. However, more changes are coming soon.

When the synthesis model is changed, there will be a detailed description of everything here.

7.2 MIDI functions

7.2.1 MIDI bank select

Control Change #32 (Bank Select LSB) has been modified to use the byBankLSB value.

7.2.2 Other MIDI functions

All other MIDI functions are unchanged from legacy SF2.04.

7.3 Parameter units, generators, modulators and NRPNs

All of these are the same as legacy SF2.04.

7.4 On implementation accuracy

E-mu was very lax when it came to accuracy of legacy SF implementations. This was because of the limitations of computers and legacy soundcards that the legacy SF spec and its implementations were initially designed to run on. While this was the only way that legacy SF could gain the popularity that it did with software implementations, this meant that bank developers often had to declare the program that their file was intended to be used with. This hampered interoperability with different SF players, including those that may have been embedded into musical instruments.

Because today's computers are much faster, and legacy soundcards are no longer in widespread use, the requirements for implementation accuracy in SFe are far more strict. All SFe players are required to recognise the flag sub-chunk (section 5.12.2) and warn the user if there is a mismatch between feature flags in the bank and the program's support.

While the flag sub-chunk will be useful to alert users about incompatible programs, program developers should make an effort to ensure that their program is 100% compatible.

Section 8

SFe error handling

8.1 Structurally Unsound errors

"Structurally Unsound" errors are those defined by E-mu (in legacy SF2.04), Werner Schweer (in Werner SF3) and the SFe Team (in SFe) to prevent the bank from working properly in a way that means that it can not be used. These errors must be fixed before an SF player can load it, unless the SF player implements SFe automatic repair.

The error correction process for structural errors in SFe is slightly different from that in legacy SF2.04:

- If a ds64 chunk, BW64 or RF64 header is found in a file, SFe players that do not support 64-bit chunk headers should output a specific error, as mentioned in the SFe program specification.
- If the ds64 chunk is missing, the bank uses a 32-bit chunk header. Make sure that ckSize is accurate, using the same techniques as for SF2.04.
 - If the value of ckSize is 4,294,967,295 (like in ds64), or any other inaccurate value, reject the file as "Structurally Unsound."
 - If the value of ckSize is correct, open the file and output a warning message, as mentioned in the SFe program specification.
 - More advanced programs may also recognize files larger than 4GiB with 32-bit headers. However, such a file should be repaired with a 64-bit header as soon as possible.
- Banks with 64-bit headers, but with a ckSize value that is not 4,294,967,295 should be rejected as "Structurally Unsound," as this is not valid in RIFF64.

8.2 Non-critical errors

"Non-critical" errors are errors that mean that data cannot be read properly, however as they are not Structurally Unsound errors, the file can be loaded. What usually happens is that the damaged data is ignored, and the rest of the bank is loaded.

While non-critical errors don't prevent the use of the bank, it is important that they are fixed to ensure that the bank functions work as intended on all SF players that conform to a specification, whether that be legacy SF2.04, Werner SF3 or SFe.

8.3 Duplicated preset locations within files

This occurs when the file is structurally damaged or manually edited in a manner where more than one preset has the same value of `byBankMSB`, `byBankLSB` and `wPreset` (for instance, `015:000:081`).

- In legacy SF2.04, the first preset in the location would be used by default, but the other presets would still be retained.
- Such other presets must be moved before they can be used.
- In SFe, the preset to be used is no longer necessarily the first one. Instead, selecting the correct preset (or combination of presets) to use will be permissible.
- Such a feature is optional, and if not implemented, the player should use the legacy SF2.04 behavior in these cases (use the first preset found).
- Editors should warn the user if such presets are found.
- This behavior might change in future versions, so please take the `ifil` value, and later versions of this specification, into account.

8.4 Duplicated preset locations across files

This occurs when multiple files are loaded simultaneously (now a required feature for SFe), but some or all of the files have presets with identical byBankMSB, byBankLSB and wPreset values.

- Behavior was undefined in legacy SF2.04.
- This was because multiple file loading was not a standard feature mandated in the legacy SoundFont® standard.
- Legacy SF2.04 and Werner SF3 compatible software developers therefore had the liberty to implement multiple file loading; however, they wanted to.
- This edge case will now be defined in SFe.
- If multiple presets across loaded files have the same value of byBankMSB, byBankLSB and wPreset, then the preset to be used may be selectable from all the presets with the same bank location (in the way described in section 8.3).
- Such a feature is also optional, and if not implemented, the player should use the legacy SF2.04 behavior in these cases (use the first preset found).
- Editors should warn the user if such presets are found.
- This behavior might change in future versions, so please take the ifil value, and later versions of this specification, into account.

8.5 Undefined chunks

Legacy SF2.04 players should ignore SFe-specific sub-chunks, as prescribed by E-mu.

Also, SFe 4.0 compatible players, which do not support future versions, should ignore sub-chunks which are used in future versions.

8.6 Unknown Enumerators

Any SFvx versions of 4.1 or above may have unknown enumeration values for an SFe 4.0 player.

This is entirely expected, and if unknown enumeration values are found, the Generator/Modulator should be ignored.

8.7 Maximum File Size Limit Exceeded

This occurs when the user loads a file that is larger than the maximum size that the SFe program can accommodate.

- In the AWE32 and AWE64, the limit was dependent on the memory installed on the card, which was up to 28 MiB.
- Later, the sound cards allowed the user to load files with file sizes up to system memory.
- SFe has defined limits that are found in the separate program specification.
- If these limits are reached, you can reject loading of the file with an error, or attempt to load the file anyway.
- SFe programs should warn the user if processing was automatically done to a file to reduce the file size to be in range.
- If multiple files are loaded, and the limit is reached, the order of files to be loaded can be defined by the author of the SFe-compatible software.

8.8 MIDI Errors

If a non-existent bank/preset combination is selected, the software should revert to the preset with `byBankMSB` and `byBankLSB` values of zero. If there is no such preset, then the software should revert to the preset with the lowest possible `byBankMSB` value, and then the `byBankLSB` value.

8.9 Illegal parameter values, out of range values, missing required items and illegal enumerators

These are handled as in legacy SF2.04.

Section 9

SiliconSFe

9.1 SiliconSFe overview

While we are unaware of any shipping products using the SiliconSF system found in SFSPEC24.PDF (the AWE cards used an early predecessor of SiliconSF), you can use ROM samples formatted in the SiliconSF format with SFe.

9.2 Header format

9.2.1 About the header format

The SiliconSFe header format is almost identical to legacy SF2.04, however an explanation is provided here due to poor documentation of SiliconSF.

Here is the SiliconSFe header format:

```
typedef struct romHdrType{
    DWORD romRiffHdr;
    DWORD romByteSize;
    CHAR romInterleaveIndex;
    CHAR romRevision[3];
    CHAR romVer[4];
    SHORT bankChecksum;
    SHORT bankChecksum2sComplement;
    CHAR bankSFeVersion;
    CHAR bankProduct[16];
    BYTE bankSampleCompType;
    CHAR filler1[2];
    CHAR bankStyle[16];
    CHAR bankCopyright[80];
    DWORD romSFeBankStart;
    DWORD romSineWaveStart;
    DWORD filler2[124];
    SHORT sampleSineWave[SINEWAVESIZE];
} romHdr;
```

9.2.2 romRiffHeader

In SiliconSFe, it is defined as the FourCC used by the chunk header type used by the integrated SF bank, for example RIFF, RF64, RIFD, etc.

In the legacy SF2.04 specification, this is named `romRsrc` and was declared by Creative as "unused". The name in SiliconSFe more accurately describes its usage.

9.2.3 romByteSize

This is an UNSIGNED DWORD value with the size of the SiliconSFe ROM blob in bytes. It is limited to 4 GiB in SiliconSFe 1.0. Signed integers are prohibited.

9.2.4 romInterleaveIndex

This is used for interleaved ROMs. You can interleave up to 256 ROMs with one SiliconSFe blob.

In the legacy SF2.04 specification, this is named `interleaveIndex`.

9.2.5 romRevision

This is a revision identifier as an integer. It is 3 bytes long and is independent from the version number found in `romVer`.

In the legacy SF2.04 specification, this is named `revision`.

9.2.6 romVer

This corresponds to the `iver` value in the integrated SF bank.

In the legacy SF2.04 specification, this is named `id` and is erroneously listed as corresponding to the `irom` value. The name in SiliconSFe more accurately describes its usage.

9.2.7 bankChecksum

This stores the CRC-16 (ARC) checksum of the integrated SF bank.

In the legacy SF2.04 specification, this is named `checksum`.

9.2.8 bankChecksum2sComplement

This stores the twos-complement of the value found in `bankChecksum`.

In the legacy SF2.04 specification, this is named `checksum2sComplement`.

9.2.9 bankSFeVersion

This value should be the same as the `wSFeSpecMajorVersion` value in the `SFvx` sub-chunk in SFe, and the same as the `wMajor` value in the `ifil` sub-chunk in non-SFe. For an unknown or other format, this value is 0.

In the legacy SF2.04 specification, this is named `bankFormat` and was declared by Creative as "unused".

9.2.10 bankProduct

This is a UTF-8 string that stores the product name (conventionally SiliconSFe).

In the legacy SF2.04 specification, this is named `product`.

9.2.11 sampleCompType

For the purpose of SiliconSFe, this value is 1 if any kind of sample precompensation is used, and 0 otherwise.

In the legacy SF2.04 specification, Creative said that it indicates the type of sample precompensation that is used in the SiliconSF blob.

9.2.12 bankStyle

This is a UTF-8 string that describes the musical style of the contents of the integrated SF bank.

In the legacy SF2.04 specification, this is named `style`.

9.2.13 bankCopyright

This is a UTF-8 string that stores copyright information about the SiliconSFe blob.

In the legacy SF2.04 specification, this is named `copyright`.

9.2.14 romSFeBankStart

This stores the location in the SiliconSFe blob where the integrated SF bank starts.

In the legacy SF2.04 specification, this is named `sampleStart`. The name in SiliconSFe more accurately describes its usage.

9.2.15 romSineWaveStart

This stores the location in the SiliconSFe blob where the test sine wave sample starts.

In the legacy SF2.04 specification, this is named `sineWaveStart`.

9.2.16 sampleSineWave

This contains SHORT values that correspond to a sine wave sample.

In the legacy SF2.04 specification, this is named `sineWave`.

9.3 AWE ROM emulator

9.3.1 Introducing the AWE ROM emulator

Except for when size concerns prohibit its inclusion, SFe players should include an AWE ROM emulator.

- The AWE ROM emulator includes 152 samples.
- The file size should be 1MB, as all samples should be to the same standard as the original.
- Samples which the program developer has the right to use can be used as a replacement for the original ROM samples.
- Freely-usable reference implementation samples are available, but are not intended for production use.
- Sample names will remain the same, but there will be acceptable alias names.
- Emulators should also be able to open up SF files (either legacy SF or SFe) containing samples and metadata.
- There may or may not be instruments or presets.

9.3.2 ROM emulator sample specification

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|--------|----------------------|--------------------------------|---------------|-------------|-------------------------------|
| 1 | acbasse1 | Acoustic Bass E1 | 1535 | 1408-1530 | |
| 2 | accordax2 | Accordion A#2 Accordian A#2 | 620 | 566-616 | Either spelling is acceptable |
| 3 | accordfx2 | Accordion F#2 Accordian F#2 | 1049 | 979-1044 | Either spelling is acceptable |
| 4 | accordfx3 | Accordion F#3 Accordian F#3 | 858 | 794-854 | Either spelling is acceptable |
| 5 | acgtrb3 | Acoustic Guitar B3 | 6241 | 6168-6236 | |
| 6 | acgtrg2 | Acoustic Guitar G2 | 4882 | 4800-4877 | |
| 7 | agogolotone | Agogo Low Tone Agogo Bell | 4467 | 7-4463 | |
| 8 | applause | Applause | 8161 | 7-8156 | |
| 9 | arcocelloax2 | Arco Cello A#2 | 847 | 799-842 | |
| 10 | arcocelld2 | Arco Cello D2 | 1200 | 1127-1195 | |
| 11 | arcoviolinc3 | Arco Violin C3 | 1767 | 1702-1762 | |
| 12 | arcoviolinc4 | Arco Violin C4 | 1634 | 1569-1629 | |
| 13 | arcovioline3 | Arco Violin E3 | 1086 | 1035-1081 | |
| 14 | arcoviolingx2 | Arco Violin G#2 | 1732 | 1691-1727 | |
| 15 | arcoviolingx3 | Arco Violin G#3 | 1075 | 1032-1070 | |
| 16 | asaxc2 | Alto Sax C2 | 1150 | 1054-1145 | |
| 17 | asaxc4 | Alto Sax C4 | 1191 | 1130-1186 | |
| 18 | asaxd3 | Alto Sax D3 | 696 | 643-691 | |
| 19 | asaxe2 | Alto Sax E2 | 1228 | 1150-1223 | |
| 20 | asaxf3 | Alto Sax F3 | 910 | 863-905 | |
| 21 | asaxg2 | Alto Sax G2 | 1639 | 1567-1634 | |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|--------|----------------------|---|---------------|-------------|--|
| 22 | bagpipedrna | Bag Drone A1 Bag Drone | 1921 | 1764-1913 | Based on name in 8MB E-mu bank |
| 23 | banjod3 | Banjo D3 | 3583 | 3540-3579 | |
| 24 | banjog2 | Banjo G2 | 2850 | 2784-2845 | |
| 25 | bassguitloop | Bass Guitar Loop | 125 | 9-120 | |
| 26 | bassoonc2 | Bassoon C2 Bassoon | 1059 | 938-1053 | Only sample of that instrument |
| 27 | bd15 | Bass Drum Kick Drum | 1603 | 7-1599 | |
| 28 | belltree | Bell Tree | 7466 | 6263-7461 | |
| 29 | bockclave | Claves | 1774 | 7-1770 | |
| 30 | brasssectc3 | Brass Section C3 | 5600 | 5533-5595 | |
| 31 | brasssectf5 | Brass Section F5 | 5581 | 4989-5035 | |
| 32 | bsawtoothwavea3 | Sawtooth Wave Sample Bass SawtoothWave A3 Bass Sawtooth Wave | 70 | 15-65 | Real time synthesis activated without "Sample" suffix |
| 33 | cabasastrk | Cabasa | 2679 | 7-2675 | Based on name in 8MB E-mu bank |
| 34 | chanterax1 | Bagpipe A#1 Bagpipe Chanter A#1 Chanter | 1858 | 1802-1850 | Based on name in 8MB E-mu bank Only sample of that instrument |
| 35 | chcrash | China Crash Cymbal | 9700 | 6162-9695 | Based on name in 8MB E-mu bank |
| 36 | clarinetb2 | Clarinet B2 | 701 | 586-695 | |
| 37 | clarinetd2 | Clarinet D2 | 677 | 596-671 | |
| 38 | clavc2 | Clav C2 Clav | 2985 | 2836-2980 | Only sample of that instrument |
| 39 | coldglass7wave | Tinkle Bell Wave 1 Tinker Bell Wave 1 | 70 | 17-65 | The alias is because of its use in the 1MB E-mu bank. Either spelling is acceptable |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|--------|----------------------|--|---------------|-------------|--|
| 40 | coldglass12wave | Tinkle Bell Wave 2 Tinker Bell Wave 2 | 91 | 15-86 | The alias is because of its use in the 1MB E-mu bank. Either spelling is acceptable |
| 41 | contraviobass | Contrabass | 1443 | 1302-1438 | |
| 42 | cowbell | Cowbell | 1760 | 7-1756 | |
| 43 | crash5 | Crash Cymbal | 13534 | 8024-13529 | |
| 44 | distgtra2 | Distortion Guitar A2 Dist Gtr A2 | 1832 | 1745-1827 | Based on name in 8MB E-mu bank |
| 45 | distgtra3 | Distortion Guitar A3 Dist Gtr A3 | 1243 | 1195-1238 | Based on name in 8MB E-mu bank |
| 46 | distgtrd4 | Distortion Guitar A4 Dist Gtr A4 | 1766 | 1593-1761 | Based on name in 8MB E-mu bank |
| 47 | distgtre3 | Distortion Guitar E3 Dist Gtr E3 | 1432 | 1372-1427 | Based on name in 8MB E-mu bank |
| 48 | dlcimrc3 | Dulcimer C3 Dulcimer | 3835 | 3778-3830 | Only sample of that instrument |
| 49 | ebongostone | H Bongo Rim | 3204 | 7-3200 | Based on name in 8MB E-mu bank |
| 50 | elguitard2 | Electric Guitar D2 Electric Guitar | 1481 | 1401-1476 | Only sample of that instrument |
| 51 | enghorndx3 | English Horn D#3 English Horn | 1540 | 1479-1534 | Only sample of that instrument |
| 52 | epiano2ms | E Piano 2 | 1173 | 1120-1168 | Only sample of that instrument |
| 53 | femalevoiceg2 | Female Voice G2 Female Voice | 8759 | 338-8755 | Only sample of that instrument |
| 54 | filtersnap | Filter Snap | 420 | 7-414 | |
| 55 | floortombrite | Floor Tom Acoustic Tom | 7172 | 5236-7167 | Based on name in 8MB E-mu bank |
| 56 | flutec4 | Flute C4 | 1432 | 1365-1427 | Only sample of that instrument |
| 57 | frenchhorn4 | French Horn G4 French Horn | 1485 | 1420-1480 | Only sample of that instrument |
| 58 | Fretlessa2 | Fretless Bass A2 Fretless Bass | 2341 | 2165-2336 | Only sample of that instrument |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|---------------|-----------------------------|--|----------------------|--------------------|--|
| 59 | glockloopc4 | Glockenspiel C4 Glockenspiel Glockenspiel Loop | 216 | 7-211 | Only sample of that instrument |
| 60 | gsbassd2 | Finger Bass D2 Finger Bass | 905 | 750-900 | Only sample of that instrument |
| 61 | guiro2 | Guilo Up | 2764 | 7-2759 | Based on name in 8MB E-mu bank |
| 62 | guirodown | Guilo Down | 2648 | 7-2644 | |
| 63 | guitar1 | Bandoneon | 140 | 69-135 | The alias is because of its use in the 1MB E-mu bank. |
| 64 | guitarfret | Guitar Fret Fret Noise | 3572 | 7-3567 | Based on name in 8MB E-mu bank |
| 65 | gunshot | Gun Shot | 5396 | 7-5392 | Based on name in 8MB E-mu bank |
| 66 | harmguitard3 | Guitar Harmonics D3 Guitar Harmonics GtrHarmonics D3 | 344 | 298-338 | Based on name in 8MB E-mu bank Only sample of that instrument |
| 67 | harmonicaa3 | Harmonica A3 Harmonica | 974 | 915-969 | Only sample of that instrument |
| 68 | harpsichordc3 | Harpsichord C3 Harpsichord | 1391 | 1294-1386 | Only sample of that instrument |
| 69 | hatopenms | Open High Hat | 11710 | 5828-11705 | Based on name in 8MB E-mu bank |
| 70 | hrmnmutec3 | Harmon Mute C3 | 1485 | 1420-1481 | |
| 71 | hrmnmutec4 | Harmon Mute C4 | 903 | 840-898 | |
| 72 | htrumpetax3 | Trumpet A#3 | 1663 | 1602-1658 | |
| 73 | htrumpetc3 | Trumpet C3 | 1653 | 1598-1648 | |
| 74 | htrumpetd2 | Trumpet D2 | 1674 | 1628-1669 | |
| 75 | htrumpetf3 | Trumpet F3 | 1497 | 1454-1492 | |
| 76 | htrumpetg2 | Trumpet G2 | 1636 | 1598-1631 | |
| 77 | jazzguitloop | Jazz Guitar Jazz Guitar Loop | 119 | 9-114 | |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|--------|----------------------|---------------------------------|---------------|-------------|--------------------------------|
| 78 | kotod3 | Koto D3 Koto | 5709 | 5666-5704 | Only sample of that instrument |
| 79 | kpianob1 | Piano B1 | 17232 | 10655-17227 | |
| 80 | kpianob4 | Piano B4 | 25217 | 15428-25212 | |
| 81 | kpianob5 | Piano B5 | 5399 | 2963-5394 | |
| 82 | kpianocx4 | Piano C#4 | 21574 | 21506-21569 | |
| 83 | kpianodx5 | Piano D#5 | 6146 | 6093-6141 | |
| 84 | kpianof5 | Piano F5 | 6980 | 6848-6974 | |
| 85 | kpianof5 #02 | Piano F5#02 | 6980 | 4148-4202 | |
| 86 | kpianog2 | Piano G2 | 22131 | 17637-22127 | |
| 87 | lefone | Telephone | 1585 | 8-1577 | Based on name in 8MB E-mu bank |
| 88 | lowtumba | Low Tumba Tone | 4022 | 7-4018 | Based on name in 8MB E-mu bank |
| 89 | maracas | Maracas | 3254 | 7-3250 | |
| 90 | marimbac3 | Marimba C3 Marimba | 817 | 788-812 | Only sample of that instrument |
| 91 | mbongotone | M Bongo Tone | 2724 | 7-2720 | Based on name in 8MB E-mu bank |
| 92 | mgtr | Muted Guitar Gtr Mute | 836 | 766-831 | Based on name in 8MB E-mu bank |
| 93 | nguitb2 | Nylon Guitar B2 N Guitar B2 | 5193 | 5125-5188 | Based on name in 8MB E-mu bank |
| 94 | nguitf2 | Nylon Guitar F2 N Guitar F2 | 3829 | 3727-3824 | Based on name in 8MB E-mu bank |
| 95 | oboeax3 | Oboe A#3 | 998 | 958-992 | |
| 96 | oboecx3 | Oboe C#3 | 892 | 832-886 | |
| 97 | oboefx3 | Oboe F#3 | 1226 | 1177-1220 | |
| 98 | Oboeresynwaved4 | Oboe Resynth D4 Oboe Resynth | 140 | 69-135 | |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|---------------|-----------------------------|---|----------------------|--------------------|--|
| 99 | ocarinafx2 | Ocarina F#2 Ocarina | 1187 | 1144-1182 | Only sample of that instrument |
| 100 | octavewave | Octave Wave | 140 | 69-135 | |
| 101 | oohvoicec3 | Ooh Voice C3 Ooh Voice | 9102 | 35-9097 | Only sample of that instrument |
| 102 | orchhit2 | Orchestra Hit Orch Hit | 4566 | 7-4562 | Based on name in 8MB E-mu bank |
| 103 | organ19d4wave | Space Voice | 140 | 69-135 | The alias is because of its use in the 1MB E-mu bank. |
| 104 | organwave | Organ Wave Organ Wave 1 | 2675 | 2614-2669 | |
| 105 | organwavea3 | Organ Wave A3 Organ Wave 2 | 2940 | 2900-2934 | |
| 106 | paisteping | Ride Cymbal Ride Ping | 13293 | 7459-13288 | Based on name in 8MB E-mu bank |
| 107 | pizzviolinc3 | Pizzicato Strings C3 Pizzicato Strings Pizzicato Violin C3 Pizzicato Violin Pizz Violin C3 Pizz Violin | 1560 | 1499-1555 | Based on name in 8MB E-mu bank Only sample of that instrument |
| 108 | pluckharp | Pluck Harp Harp | 3478 | 3409-3473 | |
| 109 | quicadown | Quica Downstroke | 1627 | 7-1623 | Based on name in 8MB E-mu bank |
| 110 | quintoslap | Quinto Slap QuintoClosedSlap | 2923 | 7-2919 | Based on name in 8MB E-mu bank |
| 111 | quintotone | Quinto Tone | 3091 | 7-3087 | Based on name in 8MB E-mu bank |
| 112 | recorderax2 | Recorder A#2 Recorder | 1360 | 1298-1352 | Only sample of that instrument |
| 113 | resynth4d4wave | Bowed Glass D4 Bowed Glass | 140 | 69-135 | Only sample of that instrument |
| 114 | rhodeschime | E Piano 1 Chime | 284 | 7-279 | |
| 115 | rideping | Ride Bell | 6034 | 3614-6029 | Based on name in 8MB E-mu bank |
| 116 | Sambawhistle | Samba Whistle | 1687 | 1121-1673 | |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|---------------|-----------------------------|--|----------------------|--------------------|---|
| 117 | sawstackwavems | Saw Stack Wave Saw Stack | 13749 | 289- 13745 | |
| 118 | sb2 | Slap Bass 2 | 2464 | 2345- 2459 | |
| 119 | scratch | Scratch | 1661 | 7-1657 | |
| 120 | shakua2 | Shakuhachi A2 Shakuhachi Siku E2 | 7468 | 2085- 7463 | Only sample of that instrument |
| 121 | sikue2 | Siku Bottle Blow E2 Bottle Blow | 5314 | 7-5310 | Only sample of that instrument |
| 122 | sinetick | Sinetick | 73 | 7-68 | |
| 123 | sinewave | Sine Wave Sample | 140 | 69-135 | Real time synthesis activated without "Sample" suffix |
| 124 | sitardx4 | Sitar D#4 Sitar | 2316 | 2275- 2311 | Only sample of that instrument |
| 125 | slapbass1c3 | Slap Bass 1 C3 Slap Bass 1 | 2121 | 1817- 2115 | Only sample of that instrument |
| 126 | snare24 | Snare | 3877 | 7-3873 | |
| 127 | squarewave | Square Wave Sample | 2427 | 2365- 2420 | Real time synthesis activated without "Sample" suffix |
| 128 | ssaxdx4 | Soprano Sax D4 Soprano Sax | 1189 | 1136- 1184 | Only sample of that instrument |
| 129 | steeldrum | Steel Drum SteelDrum | 2898 | 2857- 2891 | Based on name in 8MB E-mu bank |
| 130 | stix | Drum Stick Side Stick | 370 | 7-366 | Based on name in 8MB E-mu bank |
| 131 | stringsdx4 | Strings D#4 | 10727 | 3098- 10722 | |
| 132 | stringsf3 | Strings F3 | 8647 | 3415- 8642 | |
| 133 | stringsg2 | Strings G2 | 9309 | 2609- 9304 | |
| 134 | synthbassloop | Synth Bass Loop Synth Bass | 441 | 8-435 | |
| 135 | Synthstringsc4 | Synth Strings C4 Synth Strings | 9967 | 1272- 9962 | Only sample of that instrument |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|---------------|-----------------------------|--|----------------------|--------------------|---|
| 136 | tamborine | Tambourine Brass Tambourine Tubular Bell D4 | 3604 | 2157-3585 | Based on name in 8MB E-mu bank |
| 137 | tbelld4wave | Tubular Bell Tubular Bell Wave D4 Tubular Bell Wave | 5417 | 4829-5412 | Only sample of that instrument |
| 138 | timpani | Timpani Timp Drum | 7699 | 7079-7695 | Based on name in 8MB E-mu bank |
| 139 | triangle | Triangle Triangle Wave | 2069 | 336-2064 | Based on name in 8MB E-mu bank |
| 140 | troma3 | Trombone A3 | 1334 | 1275-1329 | |
| 141 | tromb2 | Trombone B2 | 1331 | 1232-1326 | |
| 142 | tromd4 | Trombone D4 | 1121 | 1074-1116 | |
| 143 | tromg4 | Trombone G4 | 1569 | 1504-1564 | |
| 144 | tubaax1 | Tuba A#1 Tuba | 1964 | 1831-1960 | Only sample of that instrument |
| 145 | verbclickwave | Reverb Click Wave | 1208 | 7-1204 | |
| 146 | vibese2 | Vibes E2 Vibes | 782 | 696-777 | Only sample of that instrument |
| 147 | vibraloop | Vibra Loop | 788 | 9-783 | |
| 148 | voxdodo | Doo Voice Oohs | 3059 | 2999-3054 | Based on name in 8MB E-mu bank |
| 149 | whitenoisewave | White Noise Sample | 8294 | 7-8289 | Real time synthesis activated without "Sample" suffix |
| 150 | woodblock | Wood Block Xylophone E4 Xylophone Xyo Unlooped E4 | 1164 | 7-1160 | |
| 151 | Xyloe4looped | Xylo Unlooped XylophoneUnlooped E4 Xylophone Unlooped | 980 | 7-976 | Only sample of that instrument |

| Number | Original Sample Name | Acceptable Aliases | Sample length | Loop points | Description |
|---------------|-----------------------------|--|----------------------|--------------------|--------------------------------|
| 152 | xyloE4unlooped | Xylo Looped E4 Xylo Looped Xylophone Looped E4 Xylophone Looped | 980 | 7-976 | Only sample of that instrument |

Sample specification is fixed at 44.1khz Mono with no links and tuning at 60 with no fine-tuning. You can either use discrete samples, or create your own SiliconSFe ROM containing the emulator samples.

Section 10

Compatibility information

10.1 Specification and structural compatibility

10.1.1 Legacy SF2.04 specification compatibility

SFe banks are not SoundFonts, and will not play properly on a legacy SF player! However, SFe banks are very similar to SoundFonts in terms of file structure, and should thus ideally be able to at least load on legacy SF players.

If a legacy SF player is not fully compatible with legacy SF2.04, then SFe files might not load at all.

10.1.2 INFO chunk and legacy compatibility

SFe files that use 32-bit chunk headers should be as compliant with the legacy standard as possible. SFSPEC24.PDF tells implementations to ignore additional sub-chunks in the `INFO-list` chunk, therefore it is possible to use the `INFO-list` chunk for additional data related to SFe.

If SFe files that use 32-bit chunk headers cannot be loaded on legacy soundcards, then something has went wrong.

SFe 4 does not support increased maximum length changes due to backward compatibility concerns.

10.1.3 phdr sub-chunk

This sub-chunk must contain at least two entries. Failure to do so will affect the compatibility with legacy SF players.

On legacy SF2.0x players, both `byBankMSB` and `byBankLSB` are read (as part of a larger `wBank` field), but only presets with a `byBankLSB` value of zero will be loaded.

You may also want to use the VArranger system for implementing LSB: for example, `115@ConcertGrand`. Support for the VArranger system is defined by bit 5 of leaf `00:07` in the flag sub-chunk.

Byte 7 of `byBankLSB` is reserved and should be preserved as read, and written as clear, to ensure backwards compatibility with legacy SF2.04. File editors should warn the user if byte 7 of `byBankLSB` is set.

10.1.4 Player implementation quirks

Some legacy SF2.0x players include quirks which are automatically loaded for all legacy SF banks. For example, a player may include bank translation when a reset is detected, to support standards that require the use of both bank select MSB and LSB, or may include changes to the modulator implementation to improve sound quality.

If an SFe bank uses an `isng` value of `SFe 4`, then programs must disable quirks. However, an `isng` value of `SFe 4 (quirks)` enables quirks mode. This means that the bank should be treated the same as E-mu 10K2 (SF2.01) or X-Fi (SF2.04), ensuring that banks converted from legacy SF2.0x that rely on quirks work properly.

SFe editors that encounter a value of `SFe 4 (quirks)` should overwrite such a value with `SFe 4` on save.

10.1.5 Generators and modulators

In case of compatibility issues, the shAmount and wAmount options have been kept in for SFe 4. They may be removed in future versions of SFe.

If the iver value is below 2.1024, and the isng value is equal to EMU8000 or another E-mu sound engine:

- A 12dB low pass filter is used to ensure compatibility with the original AWE32 sound processor.
- Controller sources remain amplitude based.
- Ignore the whole modulator structure if a reserved source type is found.
- The Controller Source Type #16-#20 acts identically to the Controller Source Type #0-#4.

If the SF version is 2.04 or below, ignore the whole modulator structure if a reserved source type is found.

While default modulators 1-4 are not used in SFe version 4, SFe programs must still use them for older versions:

- If the iver value is 2.04, use the SF2.04 version of the Default Modulator 2.
- If the iver value is 2.01, use the SF2.01 version of the Default Modulator 2.
- If the isng value is EMU8000, trigger legacy sound card mode.
- If the isng value is E-mu 10K1, E-mu 10K2, X-Fi, or SFe 4, do not trigger legacy sound card mode.
- If the isng value is SFe 4 (quirks), use the SF2.04 version of the Default Modulator 2 if a valid sm24 sub-chunk is found, otherwise use the SF2.01 version.

In SFe 4.0, programs should not define their own default modulators. This can cause playback issues if a SFe bank is used with a player that uses a different default modulator configuration to that of the editing software used.

Parameter units remain the same as SF2.04.

10.1.6 Error handling

Legacy SF players may halt on undefined chunks. Section 10.2 of SFSPEC21.PDF and SFSPEC24.PDF forbid the addition of sub-chunks to the `sdfa-list` chunk, but Creative/E-mu themselves decided to do it anyway when developing SF2.04 (by using the `sm24` sub-chunk).

Legacy SF players may halt on unknown enums, which goes against the legacy SF2.04 specification. However, at least some legacy soundcards do not error out on an unknown enum.

10.2 Sample compatibility

10.2.1 32-bit samples

Players fully compliant with legacy SF2.04 should be able to play files with 32-bit samples at 24-bit quality. However, these files may fail on software (such as Polyphone 2.4.x) that looks specifically for `sm24` and rejects anything else. It is therefore not recommended to use 32-bit samples with legacy SF2.0x players.

The `sm32` sub-chunk is implemented in the same way as `sm24` was by E-mu, to maximise compatibility, but due to the massive size, unpracticality and compatibility implications of 32-bit samples, we recommend that you use the `sm32` sub-chunk only with 64-bit chunk headers.

10.2.2 8-bit samples

The legacy specification instructs programs to ignore an `sm24` sub-chunk that is not paired with an `smp1` sub-chunk. Software that is compatible with legacy SF2.04, but is incompatible with SFe, will not play 8-bit samples, as legacy software cannot read the `ISFe-info` sub-chunk. Legacy sound cards will also ignore lone `sm24` chunks.

10.2.3 ROM samples

ROM samples may still be used; therefore bit 16 should not be ignored.

10.2.4 Incompatible compression formats

If an incompatible compression format is found, then the program may offer decompression of the incompatible file before use if you have permission from the original author of the compression formats. Some SF compression programs are notorious for restrictive licensing.

Section 11

Program specification

11.1 Program compatibility levels

These are guidelines to help SFe program developers write their programs. Not all of them need to be followed, but doing so will ensure that your program works with all SFe 4.0 banks.

All SFe programs must meet level 1 requirements. If they do not, guarantees that an SFe file will work on the program can not be made.

Level 1 is the level that embedded applications should be expected to include. Level 2 is the minimum 32-bit level for computers. Level 3 is the recommended 64-bit level for a good experience. Level 4 is maxed out. Requirements for levels 1-4 will increase with new versions.

If an implementation is unable to reach the layering requirements without crashing, it does not entirely meet the level.

11.1.1 File format specifications

| | Level 1 | Level 2 | Level 3 | Level 4 |
|---------------------------------|--|--|---|---|
| File size representation | Unsigned 32-bit integer | Unsigned 32-bit integer | Unsigned 64-bit integer | Unsigned 64-bit integer |
| File size limit | You must not use signed integers | You must not use signed integers | You must not use signed integers | You must not use signed integers |
| Sample streaming | System memory | Based on chunk header type | Based on chunk header type | Based on chunk header type |
| Total file size limit | System memory | System memory and disk streaming | System memory and disk streaming | System memory and disk streaming |
| Multiple files | Optional | At least 32 GiB | No limit | No limit |
| Legacy support | Full quality: SF2.01 and Werner SF3 Playback: SF2.04 | 8 or more Full quality: SF2.01 and Werner SF3 Playback: SF2.04 | 256 or more Full quality: SF2.01, SF2.04 and Werner SF3 Playback: SF2.04 | No limit Full quality: SF2.01, SF2.04 and Werner SF3 Playback: SF2.04 |
| Header support | 32-bit static Werner SF3 format Uncompressed, OGG Incompatible formats forbidden for write SF4: .sf4 SF2.0x: .sf2 | 32-bit static Werner SF3 format Uncompressed, OGG Incompatible formats forbidden for write SF4: .sf4 SF2.0x: .sf2 | 32-bit static, 64-bit static Werner SF3 format Uncompressed, OGG Incompatible formats forbidden for write SF4: .sf4 SF2.0x: .sf2 | 32-bit static, 64-bit static, RIFX Werner SF3 format Uncompressed, OGG Incompatible formats forbidden for write SF4: .sf4 SF2.0x: .sf2 |
| Sample compression | Werner SF3: .sf3 Any other uncompressed format is allowed | Werner SF3: .sf3 Any other uncompressed format is allowed | Werner SF3: .sf3 Any other uncompressed format is allowed | Werner SF3: .sf3 Any other uncompressed format is allowed |
| File extension | New chunks, feature flags | New chunks, feature flags | New chunks, feature flags | New chunks, feature flags |
| Information/ Metadata | | | | |

11.1.2 Sample specifications

| | Level 1 | Level 2 | Level 3 | Level 4 |
|---|--|--|--|--|
| Maximum sample rate | 44 100 Hz or greater | 50 000 Hz or greater | 96 000 Hz or greater | No limit |
| Sample bit depth | 16-bit or greater Ignore unsupported sdta sub-chunks | 16-bit or greater Ignore unsupported sdta sub-chunks | 24-bit or greater Ignore unsupported sdta sub-chunks | 32-bit or greater Ignore unsupported sdta sub-chunks |
| 8-bit samples | Optional | Optional | Optional | Mandatory |
| Maximum individual sample length | 16,777,216 samples or greater | 4,294,967,296 samples or greater | 4,294,967,296 samples or greater | Based on chunk header type |
| Loop point sets | 1 | 1 | 1 | 1 |
| Sample linking | Mono, Left/Right Includes ROM samples and SFe Compression | Mono, Left/Right, "Link" Includes ROM samples and SFe Compression | Mono, Left/Right, "Link" Includes ROM samples and SFe Compression | Mono, Left/Right, "Link" Includes ROM samples and SFe Compression |
| Number of channels | Mono, Stereo | Mono, Stereo | Mono, Stereo | Mono, Stereo |
| Sample name length | Display 8 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters |

11.1.3 Instrument specifications

| | Level 1 | Level 2 | Level 3 | Level 4 |
|-------------------------------------|---|--|--|--|
| Samples per instrument | 64 or greater | 4096 or greater | 16384 or greater | No limit |
| Simultaneous sample playback | 2 or greater | 32 or greater | 64 or greater | No limit |
| Modulators | All 4.0 modulators | All 4.0 modulators | All 4.0 modulators | All 4.0 modulators |
| Default modulators | All 4.0 default modulators | All 4.0 default modulators | All 4.0 default modulators | All 4.0 default modulators |
| Enumerators | All 4.0 enumerators | All 4.0 enumerators | All 4.0 enumerators | All 4.0 enumerators |
| Instrument name length | Display 8 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters |

11.1.4 Preset specifications

| | Level 1 | Level 2 | Level 3 | Level 4 |
|-------------------------------------|---|--|--|--|
| Instruments per preset | 2 or greater | 16 or greater | 64 or greater | No limit |
| Simultaneous sample playback | 4 or greater | 256 or greater Or up to polyphony limit | 512 or greater Or up to polyphony limit | No limit |
| Preset name length | Display 8 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters | Display 20 characters Write 20 characters |

11.1.5 Player specifications

| | Level 1 | Level 2 | Level 3 | Level 4 |
|--|--|--|--|---|
| Polyphony limit | 32 or greater with mono samples 16 or greater with stereo samples | 256 or greater with mono samples 128 or greater with stereo samples | 512 or greater with mono samples 256 or greater with stereo samples | No limit |
| Percussion toggle | byBankMSB/ byBankLSB bit 1 | byBankMSB/ byBankLSB bit 1 | byBankMSB/ byBankLSB bit 1 | byBankMSB/ byBankLSB bit 1 |
| Control change 0/32 (Bank select) | CC0: byBankMSB bits 2-8 CC32: byBankLSB bits 2-8 | CC0: byBankMSB bits 2-8 CC32: byBankLSB bits 2-8 | CC0: byBankMSB bits 2-8 CC32: byBankLSB bits 2-8 | CC0: byBankMSB bits 2-8 CC32: byBankLSB bits 2-8 |
| Control change 7/39/8 | | | | |
| Control change 40/10/42 | Mandatory | Mandatory | Mandatory | Mandatory |
| Control change 1/33/2/34/4/36 | CC1/33 (modulation wheel) required | CC1/33 (modulation wheel) required | Mandatory | Mandatory |
| Control change 5/37/11/43/84 | | | | |
| Control change 6/38 | GM1 data entry required | GM1 data entry required | GM1 data entry required | GM1 data entry required |
| Control change 12/13/44/45 | | | | |
| Control change 16/17/18/19 | | | | |
| Control change 48/49/50/51 | | | | |
| Control change 80/81/82/83 | | | | |
| Control change 64/66/67 | Mandatory | Mandatory | Mandatory | Mandatory |
| Control change 68/69 | Optional | Optional | Mandatory | Mandatory |

| | Level 1 | Level 2 | Level 3 | Level 4 |
|-------------------------------------|---|---|---|---|
| Control change | CC70 variation CC71 timbre | CC70 variation CC71 timbre | CC70 variation CC71 timbre | CC70 variation CC71 timbre |
| 70/71/72/73/74 | CC72 release time CC73 attack time |
| Control change | CC74 brightness CC75 decay time CC76 vibrato rate |
| 75/76/77/78/79 | CC77 vibrato depth CC78 vibrato delay |
| Control change 91/93 | Reverb and chorus | Reverb and chorus | Reverb and chorus | Reverb and chorus |
| Control change 92/94/95 | GM1 level effects unit required |
| Control change 96/97/98 | Mandatory | Mandatory | Mandatory | Mandatory |
| Control change 99/100/101 | | | | |
| Control change 120/121/122/1 | | | | |
| 23 | Mandatory | Mandatory | Mandatory | Mandatory |
| Control change 124/125/126/1 | | | | |
| 27 | | | | |
| GM1 reset | Mandatory | Mandatory | Mandatory | Mandatory |
| SMF formats | 0 and 1 | 0 and 1 | 0, 1 and 2 | Any |
| Number of channels | 16 or greater | 16 or greater | 64 or greater | No limit |
| ROM emulator | ROM sample support or emulator optional | ROM sample support or emulator optional | ROM sample support or emulator required | ROM sample support or emulator required |

11.2 Converting between legacy SF and SFe

11.2.1 Conversion from legacy SF2.04 to SFe

- Upgrade the `ifil` version in the header from `wMajor=2, wMinor=4` to `wMajor=2, wMinor=1024`.
- Overwrite the `isng` value with `SFe 4` (quirks).
- Create an `ISFe-list` sub-chunk with information: `SFty = "SFe-static", SFvx = 4, 0, Final, 0, "4.0a"`, flag corresponding to features used in the bank.

11.2.2 Conversion from SFe to legacy SF2.04

- Downgrade the ``ifil` version in the header from `wMajor=2, wMinor=1024` to `wMajor=2, wMinor=4`.
- Overwrite the `isng` value with `X-Fi`.
- Decompress the samples if the bank uses SFe Compression.
- Downsample any 32-bit samples to 24-bit, and remove the `sm32` sub-chunk.
- Convert all 8-bit samples to 16-bit by moving the sample data to the `smpl` sub-chunk.
- If the first 8 bits of either `wPreset` or `wBank` are identical to another patch, keep only the patch with bits 9-16 clear. If this is not found, keep only the last of the patches in the record. Clear bits 9-16 of all patches afterwards.
- Remove the `ISFe-list` sub-chunk.

11.2.3 Conversion from SFe to legacy SF2.01

- Downgrade the `ifil` version in the header from `wMajor=2, wMinor=1024` to `wMajor=2, wMinor=1`.
- Overwrite the `isng` value with `EMU8000` or `E-mu 10K1`.
- Decompress the samples if the bank uses SFe Compression.
- Downsample any 24-bit or 32-bit samples to 16-bit, and remove the `sm24` and `sm32` sub-chunks.
- Convert all 8-bit samples to 16-bit by moving the sample data to the `smp1` sub-chunk.
- If the first 8 bits of either `wPreset` or `wBank` are identical to another patch, keep only the patch with bits 9-16 clear. If this is not found, keep only the last of the patches in the record. Clear bits 9-16 of all patches afterwards.
- Remove the `ISFe-list` sub-chunk.

11.3 Converting between chunk header types

11.3.1 Conversion of 32-bit static to 64-bit static

- Replace RIFF with RF64.
- Add a `ds64` chunk with chunk size.
- Set `ckSize` to `FF FF FF FF`.
- Replace `sfbk` with `sfen`.

11.3.2 Conversion of 64-bit static to 32-bit static

- Make sure file size does not exceed 4GiB.
- Set `ckSize` to the value of `ds64`.
- Delete `ds64`.
- Replace RF64 with RIFF.
- Rename `sfen` to `sfbk`.

11.3.3 Conversion between 32-bit static and RIFF

- Replace RIFF with RIFX (or vice versa).
- Swap the bytes to convert between big endian and little endian.

11.4 File repair programs

11.4.1 Repairing Structurally Unsound errors

ifil sub-chunk errors

To fix ifil sub-chunk errors, file repair programs must determine the correct version number of the SFe program by inspecting the file structure. File repair programs should never simply add the current SFe specification version, as that can cause a non-critical error relating to an ifil version mismatch due to the specification version being written in ISFe-list (SFvx) and not ifil.

If the file is too damaged to determine the correct ifil version, then the repair program should repair these problems first. If, after this, a definitive SF version cannot be determined, the user should be given the option to manually enter a new SF file version. The correct data in ISFe-list should also be included if the file is an SFe bank.

Alternatively, file repair programs can allow the user to manually enter the SF file version without allowing the program to automatically determine the necessary version.

PHDR sub-chunk errors

In the case of a missing PHDR sub-chunk, the program should just reconstruct an "empty" PHDR sub-chunk. The bank will not be usable until the PHDR sub-chunk is manually re-created.

If the PHDR sub-chunk contains at least two records, but is not a multiple of 38 bytes, then this indicates a structural error in one or more records. Each record can be inspected and the structure repaired. The user should also be given the choice between repairing the records and just deleting them. Repaired records must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

Non-monotonic preset bag indices should be reordered if encountered.

The PBAG sub-chunk size is usually corrected by correcting the incorrect value(s). The correct value can be reconstructed once the preset bag indices have been reordered.

PBAG sub-chunk errors

In the case of a missing PBAG sub-chunk, the program should just reconstruct an "empty" PBAG sub-chunk. The bank will not be usable until the PBAG sub-chunk is manually re-created.

If the PHDR sub-chunk is not a multiple of 4 bytes, or if there is a mismatch between the generator or modulator indices and corresponding PGEN/PMOD sub-chunk sizes, then this indicates a structural error in one or more of the preset zones. Each zone can be inspected and the structure repaired. The user should also be given the choice between repairing the zones and just deleting them. Repaired zones must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

Non-monotonic generator or modulator indices should be reordered if encountered.

The PGEN/PMOD sub-chunk sizes are usually corrected by correcting the incorrect value(s). The correct value can be reconstructed once the preset bag indices have been reordered.

PMOD sub-chunk errors

In the case of a missing PMOD sub-chunk, the program should just reconstruct an "empty" PMOD sub-chunk. The bank may be usable, but there will be a significant quality loss until the PMOD sub-chunk is manually re-created.

If the PMOD sub-chunk is not a multiple of 10 bytes, then this indicates a structural error in one or more of the preset modulators. Each modulator can be inspected and the structure repaired. The user should also be given the choice between repairing the modulators and just deleting them. Repaired modulators must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

PGEN sub-chunk errors

In the case of a missing PGEN sub-chunk, the program should just reconstruct an "empty" PGEN sub-chunk. The bank will not be usable until the PGEN sub-chunk is manually re-created.

If the PGEN sub-chunk is not a multiple of 4 bytes, then this indicates a structural error in one or more of the preset generators. Each generator can be inspected and the structure repaired. The user should also be given the choice between repairing the generators and just deleting them. Repaired generators must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

The instrument generator value is usually corrected by correcting the incorrect value(s) by inspecting the structure and determining the correct value for the instrument generator value or terminal instrument.

INST sub-chunk errors

In the case of a missing INST sub-chunk, the program should just reconstruct an "empty" INST sub-chunk. The bank will not be usable until the INST sub-chunk is manually re-created.

If the INST sub-chunk contains at least two records, but is not a multiple of 22 bytes, then this indicates a structural error in one or more records. Each record can be inspected and the structure repaired. The user should also be given the choice between repairing the records and just deleting them. Repaired records must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

Non-monotonic instrument bag indices should be reordered if encountered.

The INST sub-chunk size is usually corrected by correcting the incorrect value(s). The correct value can be reconstructed once the instrument bag indices have been reordered.

IBAG sub-chunk errors

In the case of a missing IBAG sub-chunk, the program should just reconstruct an "empty" IBAG sub-chunk. The bank will not be usable until the IBAG sub-chunk is manually re-created.

If the IHDR sub-chunk is not a multiple of 4 bytes, or if there is a mismatch between the generator or modulator indices and corresponding IGEN/IMOD sub-chunk sizes, then this indicates a structural error in one or more of the instrument zones. Each zone can be inspected and the structure repaired. The user should also be given the choice between repairing the zones and just deleting them. Repaired zones must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

Non-monotonic generator or modulator indices should be reordered if encountered.

The IGEN/IMOD sub-chunk sizes are usually corrected by correcting the incorrect value(s). The correct value can be reconstructed once the instrument bag indices have been reordered.

IMOD sub-chunk errors

In the case of a missing IMOD sub-chunk, the program should just reconstruct an "empty" IMOD sub-chunk. The bank may be usable, but there will be a significant quality loss until the IMOD sub-chunk is manually re-created.

If the IMOD sub-chunk is not a multiple of 10 bytes, then this indicates a structural error in one or more of the instrument modulators. Each modulator can be inspected and the structure repaired. The user should also be given the choice between repairing the modulators and just deleting them. Repaired modulators must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

IGEN sub-chunk errors

In the case of a missing IGEN sub-chunk, the program should just reconstruct an "empty" IGEN sub-chunk. The bank will not be usable until the IGEN sub-chunk is manually re-created.

If the IGEN sub-chunk is not a multiple of 4 bytes, then this indicates a structural error in one or more of the instrument generators. Each generator can be inspected and the structure repaired. The user should also be given the choice between repairing the generators and just deleting them. Repaired generators must be indicated so the user can make any required adjustments. This does not have to be in the file, rather just in the user interface of the program.

The sampleID generator value is usually corrected by correcting the incorrect value(s) by inspecting the structure and determining the correct value for the sampleID generator value or terminal sampleID.

SHDR sub-chunk errors

If there is no usable sample data, the program should just reconstruct an "empty" SHDR sub-chunk. However, if there is usable sample data, the program may attempt to partially reconstruct the SHDR sub-chunk. User input will be required.

If there is no valid irom sub-chunk, the program can clean these samples if they aren't being used in the bank. If they are used by the bank, then the program should then attempt to read the sample as a non-ROM sample, and if the samples are valid, then this means that the problem is fixed. If there is no valid sample, then either a new sample must be provided by the user, or a ROM should be defined for the irom sub-chunk.

Unknown sub-chunk errors

Unknown sub-chunk errors can easily be fixed; the unknown sub-chunks are simply removed.

Compressed sample errors

In SFe Compression, all PCM samples must go before compressed samples. This is fixed by rearranging the sample data in the smp1 chunk, and then updating the shdr sub-chunk.

ckSize errors

ckSize errors are easily solved by checking the ckSize values and fixing them with the correct file sizes, and in the case of 64-bit static shunk headers, setting the ds64 chunk size to the correct value.

Incompatible chunk header errors

Incompatible chunk header errors are solved by replacing chunk headers with a compatible version, or swapping the endianness of the data.

Incompatible SFty errors

TSC-related errors are corrected by rearranging the chunks, while 8-bit samples can be converted to 16-bit.

8-bit sample errors

8-bit sample errors are corrected by attempting to repair the smpl sub-chunk. If there is no smpl sub-chunk, then an SFe player can attempt to load the file with 8-bit sample mode.

11.4.2 Repairing non-critical errors

isng sub-chunk errors

In the case of a missing isng sub-chunk, a value of SFe 4 should be written.

If it doesn't end in a zero-valued byte, then add a zero-valued byte and warn the user to verify if the value is correct.

ICRD sub-chunk errors

In the case of a missing ICRD sub-chunk, a value corresponding to the current ISO-8601 date and time should be written.

Should the ICRD sub-chunk not be a valid ISO-8601 date or time, firstly attempt to parse the current value. For example, it may be in a different language. If it can't be parsed, then write a value corresponding to the current ISO-8601 date and time.

INAM, IENG, IPRD, ICOP, ICMT or ISFT sub-chunk errors

Missing sub-chunks should be filled in intuitively. For example, for an INAM sub-chunk, the file name or Unnamed bank are good starting points.

If it doesn't end in a zero-valued byte, then add a zero-valued byte and warn the user to verify if the value is correct.

With the exception of the INAM chunk, if the sub-chunk's value cannot be faithfully copied as a string, the user should be given a chance to add the correct value in UTF-8.

irom or iver sub-chunk errors

Missing sub-chunks should not be filled in unless there are ROM samples that are linked.

smpl, sm24 and sm32 sub-chunk errors

If both the sm24 and sm32 sub-chunks are present, then these sub-chunks should be combined into 16-bit samples in one smpsub-chunk. Because endianness may not be clear in this situation, you must allow the user to select the endianness of the operation.

If there is a mismatch in the ifil version and the presence of both sm24 and sm32 sub-chunks, then allow the user to select whether they want to keep these sub-chunks. If there is only one such sub-chunk, then allow the user to update the bank to use SFe with 8-bit samples.

PHDR sub-chunk errors

Any dwLibrary, dwGenre or dwMorphology value should be cleared because current versions of SFe do not implement this yet. Presets without any zones should be given a zone named Empty preset or similar.

PBAG sub-chunk errors

These problems can easily be fixed by giving zones after the first (including the global zone) an instrument generator.

PMOD or IMOD sub-chunk errors

To fix unknown or inappropriate "link" values, they should be corrected to a value given by the user. Out-of-range modulator indexes should be fixed by letting the user select the correct modulator, or define a new modulator if it doesn't exist. Circularly linked modulators should be highlighted to allow the user to break the circular links. Modulators with the same enumerators should also be highlighted, so the enumerators can be changed by the user.

PGEN sub-chunk errors

Unknown sfGenOper values should be corrected to a value given by the user. Modulators with the same sfGenOper enumerator as another modulator should be highlighted, so it can be changed by the user. Generators and modulators in inappropriate places should also be moved with references corrected. Non-global lists that don't have an instrument generator at the end should have an instrument generator added.

INST sub-chunk errors

To fix this issue, the instrument should be deleted unless it's being used by a preset. If so, the instrument should be kept, but the user should be warned so they can fix the problem.

IBAG sub-chunk errors

These problems can easily be fixed by giving zones after the first (including the global zone) a sampleID generator.

IGEN sub-chunk errors

Unknown sfGenOper values should be corrected to a value given by the user. Modulators with the same sfGenOper enumerator as another modulator should be highlighted, so it can be changed by the user. Generators and modulators in inappropriate places should also be moved with references corrected. Non-global lists that don't have a sampleID generator at the end should have an sampleID generator added.

SHDR sub-chunk errors

If the sample rate is zero, then it should be corrected automatically to the correct value via automatic pitch detection and then highlighted for the user to verify. Bad original pitch values should be corrected to the value with automatic pitch detection. Non-zero `wSampleLink` values should be set to zero if SFe Compression is in use.

You should not edit the sample rate unless it is zero.

Undefined and unknown enum, palette value and source type errors

Anything that uses these values should either be removed or highlighted so the user can inspect it. If a value is undefined for the SFvx version, but defined for a newer version, then the program can also offer to update the version to a newer version.

Precedence errors

In this case, the user should be allowed to change the generator value or to remove the offending generator entirely.

Parameter value and padding errors

In this case, the illegal or missing parameter value should be shown to the user, so they can rectify it. Padding should automatically be added wherever possible.

Unknown chunk errors

If a chunk is undefined in the SFvx version given, but defined in a later version, then the SFvx and/or ifil versions should be updated. If it is undefined in any version, then it can safely be deleted.

Compression errors

The user should be given the choice of compression algorithms to use for the sample, and `wSampleLink` values should be set to zero.

ISFe-list chunk errors

The ISFe-list sub-chunk can be repaired, however information in the ISFe-list sub-chunk should be reconstructed.

ifil chunk errors

If features or feature flags from a newer ifil or SFvx version of SFe are found on a bank with an older declared version of SFvx, then the version should be updated to the detected version of SFvx.

Incompatible compression errors

If an incompatible compression format is supported by the SFe program, then it should be decompressed and automatically recompressed into a lossless format using SFe Compression.

wPreset value errors

If invalid wPreset values are found, then they should be highlighted by the program, and the user should have the opportunity to select a different wPreset value.

Duplicated preset location errors

Duplicated preset locations should be highlighted, and the user should have the opportunity to select a different preset value.

File size limit errors

32-bit static headers and structures can be replaced with 64-bit counterparts if the file size was found to exceed 4 GiB. Alternatively, if TSC mode is supported by the SFe program, then it can be activated by moving the sdt-a-info chunk to the end and setting the correct feature flag in the ISFe-list sub-chunk.

Feature flag errors

Feature flags can be updated to accurately reflect the features that the bank uses.

11.4.3 Manual repair

Manual repair is used to completely fix a broken SFe bank. It can be partially automated, however some parts of manual repair are not automatable, and the parts that aren't automatable are highlighted.

Manual repair is intended to be part of SFe bank development tools such as SFe editors or dedicated file repair programs. It is also suitable for situations where control over the repair process by the user is required. However, it is not suitable for SF players.

Manual repair can be used to repair all Structurally Unsound errors and non-critical errors listed in section 11.4. This is the main advantage that it holds over automatic repair.

You can use it to fix a corrupted bank. A manual repair program is very useful if you develop banks, as your data may be damaged at any time for any reason.

11.4.4 Automatic repair

Automatic repair allows SFe players to play some "Structurally Unsound" banks by automatically repairing them at load time. It consists of a partial file repair program built directly into the SFe player. With future versions of this specification, SFe players that only support a higher version of the format will also gain the ability to translate old banks to run properly.

Automatic repair is intended to be part of SFe players. It is an aid to help SFe players play banks that are otherwise "Structurally Unsound". It is not a substitute for repairing the bank using file repair programs. File repair programs should always include an option to use manual repair.

Automatic repair fixes structural defects in SFe and legacy SF2.04 banks seamlessly and transparently. It should automatically repair all Structurally Unsound errors listed in section 11.4.1 except for compressed sample errors, 8-bit sample errors and anything that requires user input.

It should also repair all non-critical errors listed in section 11.4.2 except for compression errors, wPreset value errors, file size limit errors and anything that requires user input. The repair of incompatible compression errors is optional and contingent on support for such formats.

The program developers are allowed to use any repair strategy listed in section 11.4.

Automatic repair requires that the implementation give a message when invoked. This is to encourage bank developers not to rely on the feature to implicitly define any parameters.

Automatic repair is a great method to fix issues that prevent SF banks from loading correctly, but it can't repair all defects. For example, anything that requires user input is outside of the scope of automatic repair, and should be dealt with by a file repair program or SFe editor program instead.

This feature must also never overwrite files. This is the job of a file repair program. If system memory is low, it is acceptable for automatic repair implementations to create a patched bank in temporary file storage.

You must correctly setup your banks to ensure that they run properly; do not use automatic repair to implicitly define certain parameters. If you do so, then your bank may not run properly on a legacy SF player or an SFe player that doesn't implement automatic repair.

11.5 Why these guidelines?

11.5.1 File Size Representation

- SFe is a 32-bit or 64-bit format.
- Incorrect numbers of bits will therefore result in program non-compliance.
- Signed integers are not useful because a negative file size is impossible.
- To ensure that file size limits are not arbitrarily "cut in half", signed integers are prohibited.
- Programs using signed integers to represent file size are therefore not compliant with SFe.

11.5.2 File Size Limit

- You should not impose additional file size limits in SFe programs.
- SFe files that exceed these limits may not play.
- Because compatibility is not guaranteed, such programs may not be SFe compliant.

11.5.3 Sample Streaming

- You should strive to include disk streaming support.
- This is because the file size limit is almost always greater than the system memory installed.
- System memory streaming options may be available for higher performance.

11.5.4 Total File Size Limit and Multiple Files

- Always provide space for more than one file.
- SFe allows flexibility in implementing multiple files.
- It will be standardised in the future depending on the most popular system.
- The program specification requires multiple files, to guarantee proper operation of "split bank" files.
- Not implementing multiple file operation will affect compatibility.

- Such programs therefore may not be SFe compliant.

11.5.5 Legacy Support

- SFe is designed to reconcile incompatible SF extensions.
- Werner SF3 support is required; the SFe format incorporates Werner SF3 structures (SFe Compression).
- Programs that cannot recognise Werner SF3 format (SFe Compression) may not be able to open SFe files.
- 24-bit support is optional, but if 24-bit is unsupported in an SFe player, legacy SF2.04 files must not be rejected.
- Backward compatibility with legacy SF2.0x is sacrosanct when using 32-bit chunk headers, take care when creating a program.
- "Synthfont Custom Features" support is not currently included in the specification.

11.5.6 Header Support

- The header should be RIFF and sfbk with 32-bit chunk headers to preserve legacy SF2.0x compatibility.
- 32-bit only programs should recognise 64-bit chunk headers and give an error.
- The header should be RF64 and sfen with 64-bit chunk headers.
- All programs that support 64-bit chunk headeres must also support 32-bit chunk headers.

11.5.7 Sample Compression

- Compression algorithms vary, but we recommend at least one of each lossy and lossless algorithms.
- OGG and FLAC are specified in the specification.
- Werner SF3 was designed to use OGG files, so OGG is a good start for lossy algorithms.
- Opus is an alternative that is "better" than OGG according to some people.
- FLAC and "Wavpack" are free lossless compression algorithms for sounds.
- Note that while "Wavpack" has advantages over FLAC, it is not widely supported.

11.5.8 File Extension, Structure, Information and Metadata

- The file extension is .sf4. Chunk headers must be detected.
- Do not save SFe files with an extension .sf2, as it may confuse legacy SF2.0x players. Remember that SFe files are not SoundFonts!
- Use the ifil and SFvx sub-chunks to determine version, do not use the file extension.
- For SFe, you must use the prescribed chunk sizes and limits in the specification.

11.5.9 Sample Specifications

- Level 3 requires 96kHz frequency, because it is the next standard frequency above 50kHz as specified by SF2.04.
- 88.2kHz is not recommended because it is non-standard.
- Stray sdta sub-chunks must be ignored. Erroring out on extra sdta chunks is not compliant with SF2.04.
- There must not be additional limitations to sample length, besides the file size limits.
- Sample linking features from legacy SF2.04 must be supported (with the exception of SFe Compression).

11.5.10 Instrument Specifications

- High numbers of samples per instrument are specified to ensure that the program works efficiently.
- Ideally, programs should support unlimited numbers of samples per instrument.
- You must not add restrictions to number of samples per instrument or simultaneous samples.
- Modulators are fixed and must be implemented as shown in the specification and not as in SF2.04.
- In legacy SF2.04 compatibility mode, bug compatibility can be implemented.
- When field sizes increase from 20 characters, programs should be able to display at least 64 characters.
- If it is not possible to display the entire field, use an ellipsis or similar.

11.5.11 Player Specifications

- A minimum polyphony of 256 notes is in place if possible.
- The `byBankLSB` percussion toggle is to fix the bank select LSB function.
- All control changes listed as "mandatory" must be supported.
- If it is not supported, playback may be severely impacted, resulting in a program that is not SFe compliant.
- This is already a problem with legacy SF2.0x, as not all SF2 players support modulators.
- Reset support and multiple simultaneous drum kits might not be required, but it is suggested.
- 64-channel MIDI file support is suggested, and may be required in the future for SFe.
- The ROM emulator may be required in the future, as many SFe files will make full use of it.

11.6 How to test your program with SFSpecTest

11.6.1 What does SFSpecTest do?

By using SFSpecTest by mrbumpy409 ([available here](#)), you can test your SFe player and determine which feature flags to set.

SFSpecTest was written by the author of GeneralUserGS, one of the most popular legacy SF2.0x banks, and is thus a good benchmark for legacy SF2.04 players.

Because SFe is a superset of legacy SF2.04, it is also a good tool to determine what SF2.04 features your program support, allowing you to set the correct feature flags for your program.

11.6.2 Branch 00 Foundational synthesis engine

In leaf 00:00, if your player passes SFSpecTest test #8 (Scale Tune/Root Key), you can set all four defined bits.

In leaf 00:03, set bits 1-16 to the maximum frequency attained in SFSpecTest test #9 (Initial Filter Cutoff), and bits 17-24 to the maximum resonance attained in SFSpecTest test #10 (Filter Resonance).

In leaf 00:04, if your player passes SFSpecTest test #11 (Attenuation Amount), you can set the first two defined bits. If your player passes SFSpecTest test #5a (Modulation LFO A) and #12 (Negative Attenuation Amount), you can also set the third bit.

In leaf 00:05, set bit 1 if you pass SFSpecTest test #17a (Reverb A), bit 2 if you pass SFSpecTest test #17b (Reverb B), bit 3 if you pass SFSpecTest test #17c (Reverb C), bit 9 if you pass SFSpecTest test #18a (Chorus A), bit 10 if you pass SFSpecTest test #18b (Chorus B), and bit 11 if you pass SFSpecTest test #18c (Chorus C). Set bit 4 if your reverb can be adjusted, and set bit 12 if your chorus can be adjusted.

In leaf 00:06, if your player passes SFSpecTest test #5 (Modulation LFO), you can set bit 4. If your player passes SFSpecTest test #6 (Vibrato LFO) and test #7 (Mod Wheel to LFO), you can set bit 2.

In leaf 00:07, if your player passes SFSpecTest test #1 (Volume Envelope), you can set bits 1-6. If your player passes SFSpecTest test #2 (Modulation Envelope), you can set bits 9-14. If your player passes SFSpecTest test #3 (Key Number to Decay), you can set bits 8 and 16. If your player passes SFSpecTest test #4 (Key Number to Hold), you can set bits 7 and 15.

In leaf 00:0a, set bit 3 if you pass SFSpecTest test #21 (Exclusive Class). Set bit 6 if you pass SFSpecTest #16 (Sample Offset).

11.6.3 Branch 01 Modulators and NRPN

In leaf 01:01, set bit 5 if you pass SFSpecTest test #20a (Pitch Bend A) and test #20b (Pitch Bend B). Set bit 6 if you pass SFSpecTest test #20c (Pitch Bend C).

In leaf 01:02, set bit 1 if you pass SFSpecTest test #15 (CC1 to Filter Cutoff).

In leaf 01:06, set bit 1 if you pass SFSpecTest test #13 (Velocity to Attenuation Curve), and set bit 2 if you pass SFSpecTest test #14a (Velocity to Initial Filter Cutoff Curve A) and test #14b (Velocity to Initial Filter Cutoff Curve B). Set bit 4 if you pass SFSpecTest test #15 (CC1 to Filter Cutoff). Set bit 17 if you emulate SF2.00 behaviour as shown in test #14c (Velocity to Initial Filter Cutoff Curve C), set bit 18 if you emulate SF2.01 behaviour as shown in test #14d (Velocity to Initial Filter Cutoff Curve D), and set bit 19 if you emulate SF2.04 behaviour as seen in test #14e (Velocity to Initial Filter Cutoff Curve E).

11.7 Courtesy actions

For the benefit of the SFe community, please:

- share all modifications implemented in a program under this license
- do not remove the link to the latest version of the specification

The above requirements are not required due to requirements listed by the Open Source Definition, but are good practice when redistributing the specification.

Section 12

Glossary

This glossary is broadly the same as the glossary in SFSPEC24.PDF, with these additions and changes:

- Articulation – Modulation of available parameters and usage of extra samples to produce expressive musical notes.
- AWE64 – The successor to the famous AWE32, adding features such as waveguide synthesis. Used the EMU8000 synthesizer chip, like the preceding AWE32. Available in "Value" or "Gold" versions.
- Branch – A subdivision of a tree structure containing either sub-branches or leaves that include values.
- BW64 – Broadcast Wave 64, used in the RF64 Header.
- Case-insensitive – Indicates that a UTF-8 character or string treats alphabetic characters of upper or lower case as identical.
- Case-sensitive – Indicates that a UTF-8 character or string treats alphabetic characters of upper or lower case as distinct.
- Cognitone SF4 – An incompatible modification to Werner SF3 to allow support for FLAC audio compression. Because it is considered an incompatible compression format, usage is not allowed in SFe.
- DAHDSR – Stands for Delay, attack, hold, decay, sustain, release. The six-step envelope system used in SF and SFe.
- Downloadable – legacy SF2.0x, Werner SF3 or SFe file obtained from the internet. (Old meaning referred to the obsolete ROM system)
- EMU10K1 – The successor to the EMU8000, designed by E-mu® for the Creative Labs SB Live!.
- EMU10K2 – An update to the EMU10K1, designed by E-mu® for the Creative Labs SB Audigy.
- EMU20K1 – The successor to the EMU10K2, designed by E-mu® for the Creative Labs SB X-Fi.
- EMU20K2 – An update to the EMU20K1, please refer [here](#) for information on SB X-Fi cards that include it.
- FLAC – A lossless audio compression format commonly used in open-source software. Supported by Werner SF3, but not commonly used for that purpose.
- Hold – The portion of the DAHDSR envelope after the attack portion, but before the decay portion starts.
- Leaf – A value found in a tree structure at the end of a branch.
- Legacy sound card – A Sound Blaster® (or other sound card) that uses a hardware MIDI synthesiser capable of using banks in the SoundFont® format.
- Lossless compression – Said of a compression format that retains all of its data when compressed. In terms of audio, there is no loss in quality in losslessly compressed audio.

- Lossy compression – Said of a compression format that does not retain all of its data when compressed. In terms of audio, there is a loss in quality in lossily compressed audio.
- MIDI Bank – Groups of up to 128 presets, which can be selected by the two MIDI "Bank Select" control changes (CC00 and CC32).
- OGG – See "Vorbis".
- Opus – A lossy audio compression format, slightly newer than OGG but with less wide adoption.
- Quirk – Any player-specific function that is automatically enabled and modifies the behaviour of any numeric parameters used by legacy SF2.0x, including preset locations, parameters, units, modulators or NRPNs.
- Quirks mode – A mode in an SFe-compatible player that enables the implementation quirks.
- RF64 – See "RIFF64".
- RIFF64 – A 64-bit RIFF-type format. Contrast to 32-bit versions of the RIFF format. Therefore, the maximum file size is above 4 gigabytes in size.
- RIFF-type format – Formats similar to RIFF (Resource Interchange File Format), see "RIFF" in SFSPEC24.PDF for more information.
- ROM samples – Obsolete feature used in legacy sound cards, most modern SF2 files do not use this feature.
- SB – Abbreviation of "Sound Blaster®". For example, "SB X-Fi".
- SFe – A family of enhancements to the SoundFont® 2.04 formats, unofficially created after E-mu/Creative abandoned the original format. May not be structurally compatible with legacy SF2.04.
- SFe 4 – This new specification, based on SoundFont® 2.04 and Werner SF3, with a set of new features making it more realistic. Not to be confused with the incompatible Cognitone SF4 file format.
- SFe-compatible – Indicates files, data, synthesisers, hardware or software that conform to the SFe specification.
- SFe Compression – The compression system based on Werner SF3 that SFe programs should be compliant with.
- Sound Blaster® Live! – The successor to the AWE64, which improved the synthesizer chip to the EMU10K1, supporting modulators.
- Sound Blaster® Audigy – The successor to the SB Live!, containing the EMU10K2 chip.
- Sound Blaster® X-Fi – The successor to the SB Audigy, containing the EMU20K1 or EMU20K2 chip. Supports 24-Bit SoundFont® 2 files (2.04).
- Static RIFF – Any RIFF-type format with a fixed chunk size field width, including RIFF or RIFF64. See "RIFF-type format", "RIFF" and "RIFF64".
- Synth – Abbreviation of "Synthesiser," see "Synthesiser" in SFSPEC24.PDF for more information.

- Tree structure – A structure consisting of branches and leaves.
- Vorbis – A lossy audio compression format commonly used in open-source software. The basic compression format that most Werner SF3 and SFe-compatible software should be expected to implement.
- Werner SF3 – A small upgrade to SoundFont® 2.04 created by Werner Schweer to allow an open source compression solution for SoundFont® programs. Standardised as SFe Compression.

Notes



bring music to new heights