PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06K	A2	(11) International Publication Number: WO 97/07476
		(43) International Publication Date: 27 February 1997 (27.02.97)

(21) International Application Number: PCT/US96/13154

(22) International Filing Date: 13 August 1996 (13.08.96)

(30) Priority Data:
08/514,788 14 August 1995 (14.08.95) US

(60) Parent Application or Grant
(63) Related by Continuation
US 08/514,788 (CON)
Filed on 14 August 1995 (14.08.95)

(71) Applicant *(for all designated States except US)*: CREATIVE TECHNOLOGY LTD. [SG/SG]; 67 Ayer Rajah Crescent #03-18, Singapore 139950 (SG).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: ROSSUM, David, P. [US/US]; 102 Las Lomas Drive, Aptos, CA 95003 (US). GUZEWICZ, Michael [US/US]; 4467 Open Meadow Court, San Jose, CA 95129 (US). CRAWFORD, Robert, S. [US/US]; 1753 Esperanza Court, Santa Cruz, CA 95062 (US). WILLIAMS, Matthew, F. [US/US]; 205 Serrell Avenue, Santa Cruz, CA 95065 (US). RUFFCORN, Donald, F. [US/US]; 23510 Mt. Charlie Road, Los Gatos, CA 95030 (US).

(74) Agents: HAUGHEY, Paul, C. et al.; Townsend and Townsend and Crew L.L.P., 8th floor, Two Embarcadero Center, San Francisco, CA 94111-3834 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR FORMATTING DIGITAL AUDIO DATA

(57) Abstract

An audio data format in which an instrument is described using a combination of sound samples and articulation instructions which determine modifications made to the sound sample is provided. The instruments form a first, initial layer, with a second layer having presets which can be user-defined to provide additional articulation instructions which can modify the articulation instructions at the instrument level. The articulation instructions are specified using various parameters. The present invention provides a format in which all of the parameters are specified in units which relate to a physical phenomena, and thus are not tied to any particular machine for creating or playing the audio samples. The articulation parameters include generators and modulators, which provide a connection between a real-time signal and a generator. The parameter units are specified in perceptually additive units, to make the data portable and easily edited. New units are defined to give perceptual additive parameters throughout.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

METHOD AND APPARATUS FOR FORMATTING DIGITAL AUDIO DATA

BACKGROUND OF THE INVENTION

The present invention relates to the use of digital audio data, in particular a format for storing sample-based musical sound data.

The electronic music synthesizer was invented simultaneously by a number of individuals in the early 1960's, most notably Robert Moog and Donald Buchla. The synthesizers of the 1960's and 1970's were primarily analog, although by the late 70's computer control was becoming popular.

With the advances in consumer electronics made possible by VLSI and digital signal processing (DSP), it became practical in the early 1980's to replace the fixed single cycle waveforms used in the sound producing oscillators of synthesizers with digitized waveforms. This development forked into two paths. The professional music community followed the line of "sample based music synthesizers," notably the Emulator line from E-mu Systems. These instruments contained large memories which reproduced an entire recording of a natural sound, transposed over the keyboard range and appropriately modulated by envelopes, filters and amplifiers. The low cost personal computer community instead followed the "wavetable" approach, using tiny memories and creating timbre changes on synthetic or computed sound by dynamically altering the stored waveform.

During the 1980's, another relatively low cost music synthesis technique using frequency modulation (FM) became popular first with the professional music community, later transferring to the PC. While FM was a low cost and highly versatile technology, it could not match the realism of sample based synthesis, and ultimately it was displaced by sample based approaches in professional studios.

During the same time frame, the Musical Instrument Digital Interface (MIDI) standard was devised and accepted throughout the professional music community as a standard for the realtime control of musical instrument performances. MIDI has since become a standard in the PC multimedia industry as well.

The professional sample based synthesizers expanded in their capabilities in the early 1990's, to include still more DSP. The declining cost of memory brought to the wavetable approach the ability to use sampled sounds, and soon wavetable technology and sample sound synthesis became synonymous. In the mid '90s wavetable synthesis became inexpensive enough to incorporate in mass market products. These wavetable synthesizer chips allow very good quality music synthesis at popular prices, and are currently available from a variety of vendors. While many of these chips operate from samples or wave tables stored in read only memory (ROM), a few allow the downloading of arbitrary samples into RAM memory.

The Musical Instrument Digital Interface (MIDI) language has become a standard in the PC industry for the representation of musical scores. MIDI allows for each line of a musical score to control a different instrument, called a preset. The General MIDI extension of the MIDI standard establishes a set of 128 presets corresponding to a number of commonly used musical instruments.

While General MIDI provides composers with a fixed set of instruments, it neither guarantees the nature or quality of the sounds those instruments produce, nor does it provide any method of obtaining any further variety in the basic sounds available. Various musical instrument manufacturers have produced extensions of General MIDI to allow for more variations on the set of presets. It should be clear, however, that the ultimate flexibility can only be obtained by the use of downloadable digital audio files for the basic samples.

The General MIDI standard was an attempt to define the available instruments in a MIDI composition in such a way

that composers could produce songs and have a reasonable expectation that the music would be acceptably reproduced on a variety of synthesis platforms. Clearly this was an ambitious goal; from the two operator FM synthesis chips of the early PC synthesizers, through sampled sound and "wavetable" synthesizers and even "physical modelling" synthesis, a tremendous variety of technology and capability is spanned.

When a musician presses a key on a MIDI musical instrument keyboard, a complex process is initiated. The key depression is simply encoded as a key number and "velocity" occurring at a particular instant in time. But there are a variety of other parameters which determine the nature of the sound produced. Each of the 16 possible MIDI "channels" or keyboard of sound is associated at any instant to a particular bank and preset, which determines the nature of the note to be played. Furthermore, each MIDI channel also has a variety of parameters in the form of MIDI "continuous controllers" that may alter the sound in some manner. The sound designer who authored the particular preset determined how all of these factors should influence the sound to be made.

Sound designers use a variety of techniques to produce interesting timbres for their presets. Different keys may trigger entirely different sequences of events, both in terms of the synthesis parameters and the samples which are played. Two particularly notable techniques are called layering and multi-sampling. Multi-sampling provides for the assignment of a variety of digital samples to different keys within the same preset. Using layering, a single key depression can cause multiple samples to be played.

In 1993, E-mu Systems realized the importance of establishing a single universal standard for downloadable sounds for sample based musical instruments. The sudden growth of the multimedia audio market had made such a standard necessary. E-mu devised the SoundFont® 1.0 audio format as a solution. (SoundFont® is a registered trademark of E-mu Systems, Inc.) The SoundFont® 1.0 audio format was originally introduced with the Creative Technology SoundBlaster AWE32 product using the EMU8000 synthesizer engine.

The SoundFont® audio format is designed to specifically address the concerns of wavetable (sampling) synthesis. The SoundFont® audio format differs from previous digital audio file formats in that they contain not only the digital audio data representing the musical instrument samples themselves, but also the synthesis information required to articulate this digital audio. A SoundFont® audio format bank represents a set of musical keyboards, each of which is associated with a MIDI preset. Each MIDI "preset" or keyboard of sound causes the digital audio playback of one or more appropriate samples contained within the SoundFont® audio format. When this sound is triggered by the MIDI key-on command, it is also appropriately controlled by the MIDI parameters of note number, velocity, and the applicable continuous controllers. Much of the uniqueness of the SoundFont® audio format rests in the manner in which this articulation data is handled.

The SoundFont® audio format is formatted using the "chuck" concepts of the standard Resource Interchange File Format (RIFF) used in the PC industry. Use of this standard format shell provides an easily understood hierarchical level to the SoundFont® audio format.

A SoundFont® audio format File contains a single SoundFont® audio format bank. A SoundFont® audio format bank comprises a collection of one or more MIDI presets, each with unique MIDI preset and bank numbers. SoundFont® audio format banks from two separate files can only be combined by appropriate software which must resolve preset identity conflicts. Because the MIDI bank number is included, a SoundFont® audio format bank can contain presets from many MIDI banks.

A SoundFont® audio format bank contains a number of information strings, including the SoundFont® audio format Revision Level to which the bank complies, the sound ROM, if any, to which the bank refers, the Creation Date, the Author, any Copyright Assertion, and a User Comment string.

Each MIDI preset within the SoundFont® audio format bank is assigned a unique name, a MIDI preset # and a MIDI

bank =. A MIDI preset represents an assignment of sounds to keyboard keys; a MIDI Key-On event on any given MIDI Channel refers to one and only one MIDI preset, depending on the most recent MIDI preset change and MIDI bank change occurring in the MIDI channel in question.

Each MIDI preset in a SoundFont® audio format bank comprises an optional Global Preset Parameter List and one or more Preset Layers. The global preset parameter list contains any default values for the preset layer parameters. A preset layer contains the applicable key and velocity range for the preset layer, a list of preset layer parameters, and a reference to an Instrument.

Each instrument contains an optional global instrument parameter list and one or more instrument splits. A global instrument parameter list contains any default values for the instrument layer parameters. Each instrument split contains the applicable key and velocity range for the instrument split, an instrument split parameter list and a reference to a sample. The instrument split parameter list, plus any default values, contains the absolute values of the parameters describing the articulation of the notes.

Each sample contains sample parameters relevant to the playback of the sample data and a pointer to the sample data itself.

SUMMARY OF THE INVENTION

The present invention provides an audio data format in which an instrument is described using a combination of sound samples and articulation instructions which determine modifications made to the sound sample. The instruments form a first, initial layer, with a second layer having presets which can be user-defined to provide additional articulation instructions which can modify the articulation instructions at the instrument level. The articulation instructions are specified using various parameters. The present invention provides a format in which all of the parameters are specified in units which relate to a physical phenomena, and thus are

not tied to any particular machine for creating or playing the audio samples.

Preferably, the articulation instructions include generators and modulators. The generators are articulation parameters, while the modulators provide a connection between a real-time signal (i.e., a user input code) and a generator. Both generators and modulators are types of parameters.

An additional aspect of the present invention is that the parameter units are perceptually additive. This means that when an amount specified in perceptually additive units is added to two different values of the parameter, the effect on the underlying physical value will be proportionate. In particular, percentages or logarithmically related units often have this characteristic. Certain new units are created to accommodate this, such as "time cents" which is a logarithmic measure of time used as a parameter unit herein.

The use of parameter units which are related to a physical phenomena and unrelated to a particular machine make the audio data format portable, so that it can be transferred from machine to machine and used by different people without modification. The perceptually additive nature of the parameter units allows simplified editing or modification of the timbres in an underlying music score expressed in such parameter units. Thus, the need to individually adjust particular instrument settings is eliminated, with the ability to make global adjustments at the preset level.

The modulators of the present invention are specified with four enumerators, including an enumerator which acts to transform the real-time source in order to map it into a perceptually additive format. Each modulator is specified using (1) a generator enumerator identifying the generator to which it applies, (2) an enumerator identifying the source used to modify the generator, (3) the transform enumerator for modifying the source to put it into perceptually additive form, (4) an amount indicating the degree to which the modulator will affect the generator, and (5) a source amount enumerator indicating how much of a second source will modulate the amount.

The present invention also insures that the pitch information for the audio samples is portable and editable by storing not only the original sample rate, but also the original key used in creating the sample, along with any original tuning correction.

The present invention also provides a format which includes a tag in a stereo audio sample which points to its mate. This allows editing without requiring a reference to the instrument in which the sample is used.

For a further understanding of the objects and advantages of the invention, reference should be made to the ensuing description taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a drawing of a music synthesizer incorporating the present invention;

Figs. 2A and 2B are drawings of a personal computer and memory disk incorporating the present invention;

Fig. 3 is a diagram of an audio sample structure;

Figs. 4A and 4B are diagrams illustrating different portions of an audio sample;

Fig. 5 is a diagram of a key illustrating different key input characteristics;

Fig. 6 is a diagram of a modulation wheel and pitch bend wheel as illustrative modulation inputs;

Fig. 7 is a block diagram of the instrument level and preset level incorporating the present invention;

Fig. 8 is a diagram of the RIFF file structure incorporating the present invention;

Fig. 9 is a diagram of the file format image according to the present invention;

Fig. 10 is a diagram of the articulation data structure according to the present invention;

Fig. 11 is a diagram of the modulator format;

Fig. 12 is a diagram of the audio sample format; and

Fig. 13 is a diagram illustrating the relationship of the modulator enumerators and the modulator amount.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Synthesizers and Computers

Fig. 1 illustrates a typical music synthesizer 10 which would incorporate an audio data structure according to the present invention in its memory. The synthesizer includes a number of keys 12, each of which can be assigned, for instance, to a different note of a particular instrument represented by a sound sample in the data memory. A stored note can be modified in real-time by, for instance, how hard the key is pressed and how long it is held down. Other inputs also provide modulation data, such as modulation wheels 14 and 16, which may modulate the notes.

Fig. 2A illustrates a personal computer 18 which can have an internal soundboard. A memory disk 20, shown in Fig. 2B, incorporates audio data samples according to the present invention, which can be loaded into computer 18. Either computer 18 or synthesizer 10 could be used to create sound samples, edit them, play them, or any combination.

Basic Elements of Audio Sample, Modifiers

Fig. 3 is a diagram of the structure of a typical audio sample in memory. Such an audio sample can be created by recording an actual sound, and storing it in digitized format, or synthesizing a sound by generating the digital representation directly under the control of a computer program. An understanding of some of the basic aspects of the audio sample and how it can be articulated using generators and modulators is helpful in understanding the present invention. An audio sample has certain commonly accepted characteristics which are used to identify aspects of the sample which can be separately modified. Basically, a sound sample includes both amplitude and pitch. The amplitude is the loudness of the sounds, while the pitch is the wavelength or frequency. An audio sample can have an envelope for both the amplitude and for the pitch. Examples of some typical envelopes are shown in Figs. 4A and 4B. The four aspects of the envelopes are defined as follows:

Attack. This is the time taken for the sound to reach its peak value. It is measured as a rate of

change, so a sound can have a slow or a fast attack.

Decay. This indicates the rate at which a sound loses amplitude after the attack. Decay is also measured as a rate of change, so a sound can have a fast or slow decay.

Sustain. The Sustain level is the level of amplitude to which the sound falls after decaying. The Sustain time is the amount of time spent by the sound at the Sustain level.

Release. This is time taken by the sound to die out. It is measured as a rate of change, so a sound can have a fast or slow release.

The above measurements are usually referred to as ADSR (Attack, Decay, Sustain, Release) and a sound envelope is sometimes called an ADSR envelope.

The way a key is pressed can modify the note represented by the key. Fig. 5 illustrates a key in three different positions, resting position 50, initial strike position 51 and after touch position 52.

Most keyboards have velocity-sensitive keys. The strike velocity is measured as a key is pressed from position 50 to position 51, as indicated by arrow 53. This information is converted into a number between 0 and 127 which is sent to the computer after the Note On MIDI message. In this way, the dynamic is recorded with the note (or used to modify note playback). Without this feature, all notes are reproduced at the same dynamic level.

Aftertouch is the amount of pressure exerted on a key after the initial strike. Electronic aftertouch sensors, if the keyboard is equipped with them, can sense changes in pressure after the initial strike of the key between position 51 and 52. For instance, alternating between an increase and a decrease in pressure can produce a vibrato effect. But MIDI aftertouch messages can be set to control any number of parameters, from portamento and tremolo, to those which completely change the texture of the sound. Arrow 54 indicates the release of the key which can be fast or slow.

A pitch bend wheel 62 of Fig. 6 on a synthesizer is a very useful feature. By turning the wheel while holding

down a key, the pitch of a note can be bent upwards or downwards depending on how far the wheel is turned and at what speed. Bending can be chromatic, that is to say in distinguishable semitone steps, or as a continuous glide.

A modulation control wheel 64 usually sends vibrato or tremolo information. It may be used in the form of a wheel or a joystick, though the terms "modulation wheel" is often used generically to indicate modulation.

An "LFO" is often referred to in music generation, and is a basic building block. The word "frequency" as represented in the acronym LFO (Low Frequency Oscillator) is not used to indicate pitch directly, but the speed of oscillation. An LFO is often used to act on an entire voice or an entire instrument, and it affects pitch and/or amplitude by being set to a certain speed and depth of variation, as is required in tremolo (amplitude) and vibrato (pitch).

SoundFont® Audio Format Characteristics

A SoundFont® audio format is a format of data which includes both digital audio samples and articulation instructions to a wavetable synthesizer. The digital audio samples determine what sound is being played; the articulation instructions determine what modifications are made to that data, and how these modifications are affected by the musician's performance. For example, the digital audio data might be a recording of a trumpet. The articulation data would include how to loop this data to extend the recording on a sustained note, the degree of artificial attack envelope to be applied to the amplitude, how to transpose this data in pitch as different notes were played, how to change the loudness and filtering of the sound in response to the "velocity" of a keyboard key depression, and how to respond to the musician's continuous controllers (e.g., modulation wheel) with vibrato or other modifications to the sound.

All wavetable synthesizers need some way to store this data. All wavetable synthesizers which allow the user to save and exchange sounds and articulation data need some form of file format in which to arrange this data. However, the 2.0 revision SoundFont® audio format is unique in three

specific ways: it applied a variety of techniques to allow the format to be platform independent, it is easily editable, and it is upwardly and downwardly compatible with future improvements.

The SoundFont® audio format is an interchange format. It would typically be used on a CD ROM, disk, or other interchange format for moving the underlying data from one computer or synthesizer to another, for instance. Once in a particular computer, synthesizer, or other audio processing device, it may typically be converted into a format that is not a SoundFont® audio format for access by an application program which actually plays and articulates the data or otherwise manipulates it.

Fig. 7 is a diagram showing the hierarchy of the SoundFont® audio format of the present invention. Three levels are shown, a sample level 70, an instrument level 72 and a preset level 74. Sample level 70 contains a plurality of samples 76, each with its corresponding sample parameters 78. At the instrument level, each of a plurality of instruments 80 contains at least one instrument split 82. Each instrument split contains a pointer 84 to a sample, along with, if applicable, corresponding generators 86 and modulators 88. Multiple instruments could point to the same sample, if desired.

At the preset level, a plurality of presets 88 each contain at least one preset layer 90. Each preset layer 90 contains an instrument pointer 92, along with associated generators 94 and modulators 96.

A generator is an articulation parameter, while a modulator is a connection between a real-time signal and a generator. The sample parameters carry additional information useful for editing the sample.

Generators

A generator is a single articulation parameter with a fixed value. For example, the attack time of the volume envelope is a generator, whose absolute value might be 1.0 seconds.

While the list of SoundFont® audio format generators is arbitrarily expandable, a basic list follows. Appendix II contains a list and brief description of the revision 2.0 SoundFont® audio format generators. The basic pitch, filter cutoff and resonance, and attenuation of the sound can be controlled. Two envelopes, one dedicated to control of volume and one for control of pitch and/or filter cutoff are provided. These envelopes have the traditional attack, decay, sustain, and release phases, plus a delay phase prior to attack and a hold phase between attack and decay. Two LFOs, one dedicated to vibrato and one for additional vibrato, filter modulation, or tremolo are provided. The LFOs can be programmed for depth of modulation, frequency, and delay from key depression to start. Finally, the left/right pan of the signal, plus the degree to which it is sent to the chorus and reverberation processors is defined.

Five kinds of generator Enumerators exist: Index Generators, Range Generators, Substitution Generators, Sample Generators, and Value Generators.

An index generator's amount is an index into another data structure. The only two index generators are instrument and sampleID.

A range generator defines a range of note-on parameters outside of which the layer or split is undefined. Two range generators are currently defined, keyRange and kelRange.

Substitution generators are generators which substitute a value for a note-on parameter. Two substitution generators are currently defined, overridingKeyNumber and overridingVelocity.

Sample generators are generators which directly affect a sample's properties. These generators are undefined at the layer level. The currently defined sample generators are the eight address offset generators and the sampleModes generator.

Value generators are generators whose value directly affects a signal processing parameter. Most generators are value generators.

Modulators

An important aspect of realistic music synthesis is the ability to modulate instrument characteristics in real time. this can be done in two fundamentally different ways. First, signal sources within the synthesis engine itself, such as low frequency oscillators (LFOs) and envelope generators can modulate the synthesis parameters such as pitch, timbre, and loudness. But also, the performer can explicitly modulate these sources, usually by means of MIDI Continuous Controllers (Ccs).

The revision 2.0 SoundFont® audio format provides tremendous flexibility in the selection and routing of modulation by the use of the modulation parameters. A modulator expresses a connection between a real-time signal and a generator. For example, sample pitch is a generator. A connection from a MIDI pitch wheel real-time bipolar continuous controller to sample pitch at one octave full scale would be a typical modulator. Each modulation parameter specifies a modulation signal source, for example a particular MIDI continuous controller, and a modulation destination, for example a particular SoundFont® audio format generator such as filter cutoff frequency. The specified modulation amount determines to what degree (and with what polarity) the source modulates the destination. An optional modulation transform can non-linearly alter the curve or taper of the source, providing additional flexibility. Finally, a second source (amount source) can be optionally specified to be multiplied by the amount. Note that if the second source enumerator specifies a source which is logically fixed at unity, the amount simply controls the degree of modulation.

Modulators are specified using five numbers, as illustrated in Fig. 11. The relationships between these numbers are illustrated in Fig. 13. The first number is an enumerator 140 which specifies the source and format of the real-time information associated with the modulator. The second number is an enumerator 142 specifying the generator parameter affected by the modulator. The third number is a second source (amount source) enumerator 146, but this

specifies that this source varies the amount that the first source affects the generator. The fourth number 144 specifies the degree to which the second source affects the first source 140. The fifth number is an enumerator 148 specifying a transformation operation on the first source.

The revision 1.0 SoundFont® audio format used enumerators for the generators only. As new generators and modulators are established and implemented, software not implementing these new features will not recognize their enumerators. If the software is designed to simply ignore unknown enumerators, bidirectional compatibility is achieved.

By using the modulator scheme extremely complex modulation engines can be specified, such as those used in the most advanced sampled sound synthesizers. In the initial implementation of revision 2.0 SoundFont® audio format, several default modulators are defined. These modulators can be turned off or modified by specifying the same Source, Destination and Transform with zero or non-default Modulation Amount parameters.

The modulator defaults include the standard MIDI controllers such as Pitch Wheel, Vibrato Depth, and Volume, as well as MIDI Velocity control of loudness and Filter Cutoff.

The SoundFont® Audio Format Sample Parameters

The sample parameters represented in revision 2.0 SoundFont® audio format carry additional information which is not expressly required to reproduce the sound, but is useful in further editing the SoundFont® audio format bank. Fig. 12 is a diagram of the Sample Format. The original sample rate 149 of the sample and pointers to the sample Start 150, Sustain Loop Start 152, Sustain Loop End 154, and sample End 156 data points are contained in the sample parameters. Additionally, the Original Key 158 of the sample is specified in the sample parameters. This indicates the MIDI key number to which this sample naturally corresponds. A null value is allowed for sounds which do not meaningfully correspond to a MIDI key number. Finally, a Pitch Correction 160 is included in the sample parameters to allow for any mistuning that might

be inherent in the sample itself. Also, a stereo indicator 162 and link tag 164, discussed below, are included.

SoundFont® Audio Format

The SoundFont® audio format, in a manner analogous to character fonts, enables the portable rendering of a musical composition with the actual timbres intended by the performer or composer. The SoundFont® audio format is a portable, extensible, general interchange standard for wavetable synthesizer sounds and their associated articulation data.

A SoundFont® audio format bank is a RIFF file containing header information, 16 bit linear sample data, and hierarchically organized articulation information about the MIDI presets contained within the bank. The RIFF file structure is shown in Fig. 8. Parameters are specified on a precisely defined, perceptual relevant basis with adequate resolution to meet the best rendering engines. The structure of the SoundFont® audio format has been carefully designed to allow extension to arbitrarily complex modulation and synthesis networks.

Fig. 9 shows the file format image for the RIFF file structure of Fig. 8. Appendix I sets forth a description of each of the structures of Fig. 9.

Fig. 10 illustrates the articulation data structure according to the present invention. Preset level 74 is illustrated as three columns showing the preset headers 100, the preset layer indices 102, and the preset generators and modulators 104. In the example shown, a preset header 106 points to a single generator index and modulator index 108 in preset layer index 102. In another example, a preset header 110 points to two indices 112 and 114. Different preset generators can be used, as illustrated by layer index 108 pointing to a generator and amount 116 and a generator and instrument index 118. Index 112, on the other hand, only points to a generator and amount 120 (a global preset layer).

Instrument level 72 is accessed by the instrument index pointers in preset generators 104. The instrument level includes instrument headers 122 which point to instrument

split indices 124. One or more split indices can be assigned to any one instrument header. The instrument split indices, in turn, point to a particular instrument generators 126. The generators can have just a generator and amount (thus being a global split), such as instrument generator 128, or can include a pointer to a sample, such as instrument generator 130. Finally, the instrument generators point to the audio sample headers 132. The audio sample headers provide information about the audio sample and the audio sample itself.

Unit Definitions

There are a variety of specific units cited in this document. Some of these units are conventional within the music and sound industry. Others have been created specifically for the present invention. The units have two basic characteristics. First, all the units are perceptually additive. The primary units used are percentages, decibels (dB) and two newly defined units, absolute cents (as opposed to the well-known musical cents measuring pitch deviation) and time cents.

Second, the units either have an absolute meaning related to a physical phenomena, or a relative meaning related to another unit. Units in the instrument or sample level frequently have absolute meaning, that is they determine an absolute physical value such as Hz. However, in the preset level the same SoundFont® audio format parameter will only have a relative meaning, such as semitones of pitch shift.

Relative Units

Centibels: Centibels (abbreviated Cb) are a relative unit of gain or attenuation, with ten times the sensitivity of decibels (dB). For two amplitudes A and B, the Cb equivalent gain change is:

$$Cb = 200 \log10 (A/B);$$

A negative Cb value indicates A is quieter than B. Note that depending on the definition of signals A and B, a positive number can indicate either gain or attenuation.

Cents: Cents are a relative unit of pitch. A cent is 1/1200 of an octave. For two frequencies F and G, the cents of pitch change is expressed by:

cents = 1200 log2 (F/G);

A negative number of cents indicates that frequency F is lower than frequency G.

TimeCents: TimeCents are a new defined unit which are a relative unit of duration, that is a relative unit of time. For two time periods T and U, the TimeCents of time change is expressed by:

timecents = 1200 log2 (T/U);

A negative number of timecents indicates that time T is shorter than time U. The similarity of TimeCents to cents is obvious from the formula. TimeCents is a particularly useful unit for expressing envelope and delay times. It is a perceptually relevant unit, which scales with the factor as cents. In particular, if the waveform pitch is varied in cents and the envelope time parameters in TimeCents, the resulting waveform will be invariant in shape to an additive adjustment of a positive offset to pitch and a negative adjustment of the same magnitude to all time parameters.

Percentage: Tenths of percent of Full Scale is another useful relative (and absolute) measure. The Full Scale unit can be dimensionless, or be measured in dB, cents, or timecents. A relative value of zero indicates that there is no change in the effect; a relative value of 1000 indicates the effect has been increased by a full scale amount. A relative value of -1000 indicates the effect has been decreased by a full scale amount.

Absolute Units

All parameters have been specified in a physically meaningful and well-defined manner. In previous formats, including SoundFont® audio format, some of the parameters have been specified in a machine dependent manner. For example, the frequency of a low frequency modulation oscillator (LFO) might have previously been expressed in arbitrary units from 0 to 255. In revision 2.0 SoundFont® audio format, all units are specified in a physically referenced form, so that the

LFO's frequency is expressed in cents (a cent is a hundredth of a musical semitone) relative to the frequency of the lowest key on the MIDI keyboard.

When specifying any of these units absolutely, a reference is required.

Centibels: In revision 2.0 SoundFont® audio format, this is generally a "full level" note for centibel units. A value of 0 Cb for a SoundFont® audio format parameter indicates that the note will come out as loud as the instrument designer has designated for a note of "full" loudness.

TimeCents: Absolute timecents are given by the formula:

$$\text{absolute timecents} = 1200\log_2(t), \text{ where } t = \text{time in seconds}$$

In revision 2.0 SoundFont® audio format, the TimeCents absolute reference is 1 second. A value of zero represents a 1 second time or 1 second for a full (96 dB) transition.

Absolute Cents: All units of frequency are in "Absolute Cents." Absolute Cents are defined by the MIDI key number scale, with 0 being the absolute frequency of MIDI key number 0, or 8.1758 Hz. Revision 2.0 SoundFont® audio format parameter units have been designed to allow specification equal or beyond the Minimum Perceptible Difference for the parameter. The unit of a "cent" is well known by musicians as 1/100 of a semitone, which is below the Minimum Perceptible Difference of frequency.

Absolute Cents are used not only for pitch, but also for less perceptible frequencies such as Filter Cutoff Frequency. While few synthesis engines would support filters with this accuracy of cutoff, the simplicity of having a single perceptual unit of frequency was chosen as consistent with the revision 2.0 SoundFont® audio format philosophy. Synthesis engines with lower resolutions simply round the specified Filter Cutoff Frequency to their nearest equivalent.

Reproducability of SoundFont® Audio Format

The precise definition of parameters is important so as to provide for reproducability by a variety of platforms. Varying hardware platforms may have differing capabilities,

but if the intended parameter definition is known, appropriate translation of parameters to allow the best possible rendition of the SoundFont® audio format on each platform is possible.

For example, consider the definition of Volume Envelope Attack Time. This is defined in revision 2.0 SoundFont® audio format as the time from when the Volume Envelope Delay time expires until the Volume Envelope has reached its peak amplitude. The attack shape is defined as a linear increase in amplitude throughout the attack phase. Thus the behavior of the audio within the attack phase is completely defined.

A particular synthesis engine might be designed without a linear amplitude increase as a physical capability. In particular, some synthesis engines create their envelopes as sequences of constant dB/sec ramps to fixed dB endpoints. Such a synthesis engine would have to simulate a linear attack as a sequence of several of its native ramps. The total elapsed time of these ramps would be set to the attack time, and the relative heights of the ramp endpoints would be set to approximate points on the linear amplitude attack trajectory. Similar techniques can be used to simulate other revision 2.0 SoundFont audio format parameter definitions when so required.

Perceptually Additive Units

All the revision 2.0 SoundFont® audio format units which can be edited are expressed in units that are "perceptually additive." Generally speaking, this means that by adding the same amount to two different values of a given parameter, the perception will be that the change in both cases will be of the same degree. Perceptually additive units are particularly useful because they allow editing or alteration of values in an easy manner.

The property of perceptual additivity can be strictly defined as follows. If the measurement units of a perceivable phenomenon in a particular context are perceptually additive, then for any four measured values W, X, Y, and Z, where W = D+X, and Y = D+Z (D being constant), the perceived difference from X to W will be same as the perceived difference from Z to Y.

For most phenomena which can be perceived over a wide range of values perceptually additive units are typically logarithmic. When a logarithmic scale is used, the following relationships hold:

Value	Value expressed as power of ten	Log(Value)
0.1	10^{-1}	-1.0
1	10^{0}	0.0
10	10^{1}	1.0
100	10^{2}	2.0
1000	10^{3}	3.0

Thus the logarithm of 0.1 is -1, and the logarithm of 100 is 2. As can be seen, adding the same value of, for example, 1 to each log(value) increases the underlying value in each case by ten times.

If we attempt to determine, for example, perceptually additive units of sound intensity, we find that these are logarithmic units. A common logarithmic unit of sound intensity is the decibel (dB). It is defined as ten times the logarithm to the base 10 of the ratio of intensity of two sounds. By defining one sound as a reference, an absolute measure of sound intensity may also be established. It can be experimentally verified that the perceived difference in loudness between a sound at 40 decibels and one at 50 decibels is indeed the same as the perceived difference between a sound at 80 dB and one at 90 dB. This would not be the case if the sound intensity were measured in the CGS physical units of ergs per cubic centimeter.

Another perceptually additive unit is the measurement of pitch in musical cents. This is easily seen by recalling that a musical cent is 1/100 of a semitone, and a semitone is 1/12 of an octave. An octave is, of course, a logarithmic measure of frequency implying a doubling. Musicians will easily recognize that transposing a sequence of notes by a fixed number of cents, semitones, or octaves changes all the pitches by a perceptually identical difference, leaving the melody intact.

One SoundFont® audio format unit which is not strictly logarithmic is the measure of degree of reverberation or chorus processing. The units of these generators are in

terms of a percentage of the total amplitude of the sound to be sent to the associated processor. However, it is true that the perceived difference between a sound with 0% reverberation and one with 10% reverberation is the same as the difference between one with 90% reverberation and one with 100% reverberation. The reason for this deviation from strict logarithmic relationship (we might have expected the difference between 1% and 2% to be the same as 50% and 100% had the perceptually additive units been logarithmic) is that we are comparing the degree of reverberation against the full level of the direct or unprocessed sound.

Since time is typically expressed in linear units such as seconds, the present invention provides a new measure of time called "time cents," defined above on a logarithmic scale. When phenomena such as the attack and decay of musical notes are perceived, time is perceptually additive in a logarithmic scale. It can be seen that this corresponds, like intensity and pitch, to a proportionate change in the value. In other words, the perceived difference between 10 milliseconds and 20 milliseconds is the same as that between one second and two seconds; they are both a doubling.

For example, Envelope Decay Time is measured not in seconds or milliseconds, but in timecents. An absolute timecent is defined as 1200 times the base 2 logarithm of the time in seconds. A relative timecent is 1200 times the base 2 logarithm of the ratio of the times.

Specification of Envelope Decay Time in timecents allows additive modification of the decay time. For example, if a particular instrument contained a set of Instrument Splits which spanned Envelope Decay Times of 200 msec at the low end of the keyboard and 20 msec at the high end, a preset could add a relative timecent representing a ratio of 1.5, and produce a preset which gave a decay time of 300 msec at the low end of the keyboard and 30 msec at the high end. Furthermore, when MIDI Key Number is applied to modulate Envelope Decay Time, it is appropriate to scale by an equal ratio per octave, rather than a fixed number of msec per octave. This means that a fixed number of timecents per MIDI

Key Number deviation are added to the default decay time in timecents.

The units chosen are all perceptually additive. This means that when a relative layer parameter is added to a variety of underlying split parameter, the resulting parameters are perceptually spaced in the same manner as in the original instrument. For example, if volume envelope attack time were expressed in milliseconds, a typical keyboard might have very quick attack times of 10 msec at the high notes, and slower attack times of 100 msec on the low notes. If the relative layer were also expressed in the perceptually non-additive milliseconds, an additive value of 10 msec would double the attack time for the high notes while changing the low notes by only ten percent. Revision 2.0 SoundFont® audio format solves this particular dilemma by inventing a logarithmic measure of time, dubbed "TimeCents", which is perceptually additive.

Similar units (cents, dB, and percentages) have been used throughout revision 2.0 SoundFont® audio format. By using perceptually additive units, revision 2.0 SoundFont® audio format provides the ability to customize an existing "instrument" by simply adding a relative parameter to that instrument. In the example above, the attack time was extended while still maintaining the characteristic attack time relationship over the keyboard. Any other parameter can be similarly adjusted, thus providing particularly easy and efficient editing of presets.

Pitch of Sample

A unique aspect of revision 2.0 SoundFont® audio format is the manner in which the pitch of the sampled data is maintained. In previous formats, two approaches have been taken. In the simplest approach, a single number is maintained which expresses the pitch shift desired at a "root" keyboard key. This single number must be computed from the sample rate of the sample, the output sample rate of the synthesizer, the desired pitch at the root key, and any tuning error in the sample itself.

In other approaches, the sample rate of the sample is maintained as well as any desired pitch correction. When the "root" key is played, the pitch shift is equal to the ratio of the sample rate of the sample to the output sample rate, altered by any correction. Corrections due to sample tuning errors as well as those deliberately required to create a special effect are combined.

Revision 2.0 SoundFont® audio format maintains for each sample not only the sample rate of the sample but also the original key which corresponds to the sound, any tuning correction associated with the sample, and any deliberate tuning change (the deliberate tuning change is maintained at the instrument level). For example, if a 44.1 Khz sample of a piano's middle C was made, the number 60 associated with MIDI middle C would be stored as the "original key" along with 44100. If a sound designer determined that the recording were flat by two cents, a two cent positive pitch correction would also be stored. These three numbers would not be altered even if the placement of the sample in the SoundFont audio format was not such that the keyboard middle C played the sample with no shift in pitch. SoundFont audio format maintains separately a "root" key whose default value is this natural key, but which can be changed to alter the effective placement of the sample on the keyboard, and a coarse and fine tuning to allow deliberate changes in pitch.

The advantage of such a format comes when a SoundFont® audio format is to be edited. In this case, even if the placement of the sample is altered, when the sound designer goes to use the sample in another instrument, the correct sample rate (indicating natural bandwidth), original key (indicating the source of the sound) and pitch correction (so that he need not again determine the exact pitch) are available.

Revision 2.0 SoundFont® audio format provides for an "unpitched" value (conventionally -1) for the original key to be used when the sound does not have a musical pitch.

Stereo Tags

In other approaches, the sample rate of the sample is maintained as well as any desired pitch correction. When the "root" key is played, the pitch shift is equal to the ratio of the sample rate of the sample to the output sample rate, altered by any correction. Corrections due to sample tuning errors as well as those deliberately required to create a special effect are combined.

Revision 2.0 SoundFont® audio format maintains for each sample not only the sample rate of the sample but also the original key which corresponds to the sound, any tuning correction associated with the sample, and any deliberate tuning change (the deliberate tuning change is maintained at the instrument level). For example, if a 44.1 Khz sample of a piano's middle C was made, the number 60 associated with MIDI middle C would be stored as the "original key" along with 44100. If a sound designer determined that the recording were flat by two cents, a two cent positive pitch correction would also be stored. These three numbers would not be altered even if the placement of the sample in the SoundFont audio format was not such that the keyboard middle C played the sample with no shift in pitch. SoundFont audio format maintains separately a "root" key whose default value is this natural key, but which can be changed to alter the effective placement of the sample on the keyboard, and a coarse and fine tuning to allow deliberate changes in pitch.

The advantage of such a format comes when a SoundFont® audio format is to be edited. In this case, even if the placement of the sample is altered, when the sound designer goes to use the sample in another instrument, the correct sample rate (indicating natural bandwidth), original key (indicating the source of the sound) and pitch correction (so that he need not again determine the exact pitch) are available.

Revision 2.0 SoundFont® audio format provides for an "unpitched" value (conventionally -1) for the original key to be used when the sound does not have a musical pitch.

Stereo Tags

Another unique aspect of revision 2.0 SoundFont® audio format is the way in which stereo samples are handled. Stereo samples are particularly useful when reproducing a musical instrument which has an associated sound field. A piano is a good example. The low notes of a piano appear to come from the left, while the high notes come from the right. The stereo samples also add a spacious feel to the sound which is missing when a single monophonic sample is used.

In previous formats, special provisions are made in the equivalent of the instrument level to accommodate stereo samples. In revision 2.0 SoundFont® audio format, the sample itself is tagged as stereo (indicator 162 in Fig. 12), and has the location of its mate in the same tag (tag 164 in Fig. 12). This means that when editing the SoundFont audio format, a stereo sample can be maintained as stereo without needing to refer to the instrument in which the sample is used.

The format can also be expanded to support even greater degrees of sample associativity. If a sample is simply tagged as "linked", with a pointer to another member of the linked set which are all similarly linked in a circular manner, then triples, quads, or even more samples can be maintained for special handling.

Use of Identical Data to Eliminate Interpolator Incompatibility

Wavetable synthesizers typically shift the pitch of the audio sample data they are playing by a process known as interpolation. This process approximates the value of the original analog audio signal by performing mathematics on some number of known sample data points surrounding the required analog data location.

An inexpensive, yet somewhat flawed method of interpolation is equivalent to drawing a line between the two proximal data points. This method is termed "linear interpolation." A more expensive and audibly superior method instead computes a curved function using N proximal data points, appropriately dubbed N point interpolation.

Because both these methods are commonly in use, any format which purports to be portable among both types of systems must perform adequately in both. While the quality of

the data will be forced to be identical with virtually no disruption of the original data.

Mathematically stated, if X_s is the sample data point at the start of the loop, X_e is the sample data point at the loop end, and the sample rate is 50 kHz, then we can form the loop correction signal L_n:

For n from -253 to -5: $L_n = (254+n)\ (X_{(s+n)} + X_{(e+n)})/500$
For n from -4 to 4: $L_n = (X_{(s+n)} + X_{(e+n)})/2$
For n from 5 to 253: $L_n = (254-n)\ (X_{(s+n)} + X_{(e+n)})/500$

The cross-fade is similarly performed around both loop start and loop end:

For n from -253 to -5:
$X'_{(s+n)} = (245+n)\ L_n/250 + (-4-n)X_{(s+n)}/250$
For n from -4 to 4: $X'_{(s+n)} = L_n$
For n from 5 to 253:
$X'_{(s+n)} = (254-n)\ L_n/250 + (-4+n)X_{(s+n)}/250$
For n from -253 to -5:
$X'_{(e+n)} = (254+n)\ L_n/250 + (-4-n)X_{(e+n)}/250$
For n from -4 to 4: $X'_{(e+n)} = L_n$
For n from 5 to 253:
$X'_{(e+n)} = (254-n)\ L_n/250 + (-4+n)X_{(e+n)}/250$

It should be clear from the mathematical equations that the functions can be simplified by combining the averaging and cross-fading operations.

As will be understood by those familiar with the art, the present invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. For example, other units that are perceptually additive could be used rather than the ones set forth above. For example, time could be expressed as a logarithmic value multiplied by something other than 1200, or could be expressed in percentage form. Accordingly, the foregoing description is intended to be illustrative of the invention, and reference should be made to the following claims for an understanding of the scope of the invention.

APPENDIX I

4 SoundFont 2 RIFF File Format

4.1 SoundFont 2 RIFF File Format Level 0

```
<SFBK-form>   -> RIFF   'bk'          ; RIFF form header
                        {
                        <INFO-list>   ; Supplemental Information
                        <sdta-list>   ; The Sample Binary Data
                        <pdta-list>   ; The Preset, Instrument, and Sample Header data
                        }
                 )
```

4.2 SoundFont 2 RIFF File Format Level 1

```
<INFO-list>   -> LIST ('INFO'
                        {
                        <ifil-ck>     ; Refers to the version of the Sound Font RIFF file
                        <isng-ck>     ; Refers to the target Sound Engine
                        <INAM-ck>     ; Refers to the Sound Font Bank Name
                        [<irom-ck>]   ; Refers to the Sound ROM Name
                        [<iver-ck>]   ; Refers to the Sound ROM Version
                        [<ICRD-ck>]   ; Refers to the Date of Creation of the Bank
                        [<IENG-ck>]   ; Sound Designers and Engineers for the Bank
                        [<IPRD-ck>]   ; Product for which the Bank was intended
                        [<ICOP-ck>]   ; Contains any Copyright message
                        [<ICMT-ck>]   ; Contains any Comments on the Bank
                        [<ISFT-ck>]   ; The SoundFont tools used to create and alter the bank
                        }
                 )

<sdta-ck>     -> LIST ('sdta'
                        {
                        [<smpl-ck.]   ; The Digital Audio Samples
                        }
                 )

<pdta-ck>     -> LIST ('pdta'
                        {
                        <phdr-ck>     ; The Preset Headers
                        <pbag-ck>     ; The Preset Index list
                        <pmod-ck>     ; The Preset Modulator list
                        <pgen-ck>     ; The Preset Generator list
                        <inst-ck>     ; The Instrument Names and Indicies
                        <ibag-ck>     ; The Instrument Index list
                        <imod-ck>     ; The Instrument Modulator list
                        <igen-ck>     ; The Instrument Generator list
                        <shdr-ck>     ; The Sample Headers
                        }
                 )
```

4.3 SoundFont 2 RIFF File Format Level 2

```
<ifil-ck>    -> ifil(<iver-rec>)              ; e.g. 2.00
<isng-ck>    -> isng(szSoundEngine:ZSTR)      ; e.g. "EMU8000"
<irom-ck>    -> irom(szROM:ZSTR)              ; e.g. "1MGM"
<iver-ck>    -> iver(<iver-rec>)              ; e.g. 2.08
<INAM-ck>    -> INAM(szName:ZSTR)             ; e.g. "General MIDI"
<ICRD-ck>    -> ICRD(szDate:ZSTR)             ; e.g. "July 15, 1995"
<IENG-ck>    -> IENG(szName:ZSTR)             ; e.g. "John Q. Engineer"
<IPRD-ck>    -> IPRD(szProduct:ZSTR)          ; e.g. "SBAWE32"
<ICOP-ck>    -> ICOP(szCopyright:ZSTR)        ; e.g. "Copyright (c) 1995 E-mu Systems, Inc."
<ICMT-ck>    -> ICMT(szComment:ZSTR)          ; e.g. "This is a comment"
<ISTF-ck>    -> ISFT(szTools:ZSTR)            ; e.g. "Preditor 2.00a:Preditor 2.00a"

<smpl-ck>    -> smpl(<sample:WORD>)           ; 16 bit Linearly Coded Digital Audio Data

<phdr-ck>    -> phdr(<phdr-rec>)
<pbag-ck>    -> pbag(<pbag-rec>)
<pmod-ck>    -> pmod(<pmod-rec>)
<pgen-ck>    -> pgen(<pgen-rec>)
<inst-ck>    -> inst (<inst -rec>)
<ibag-ck>    -> ibag(<ibag-rec>)
<imod-ck>    -> imod(<imod-rec>)
<igen-ck>    -> igen(<igen-rec>)
<shdr-ck>    -> shdr(<shdr-rec>)
```

4.4 SoundFont 2 RIFF File Format Level 3

```
<iver-rec>   -> struct sfVersionTag
     {
     WORD wMajor;
     WORD wMinor;
     };

<phdr-rec>   -> struct sfPresetHeader
     {
     CHAR achPresetName[20];
     WORD wPreset;
     WORD wBank;
     WORD wPresetBagNdx;
     DWORD dwLibrary;
     DWORD dwGenre;
     DWORD dwMorphology;
```

```
    };

<pbag-rec>  ->  struct sfPresetBag
    {
    WORD wGenNdx;
    WORD wModNdx;
    };

<pmod-rec> ->  struct sfModList
    {
    SFModulator sfModSrcOper;
    SFGenerator sfModDestOper;
    SHORT modAmount;
    SFModulator sfModAmtSrcOper;
    SFTransform sfModTransOper;
    };

<pgen-rec>  ->  struct sfGenList
    {
    SFGenerator sfGenOper;
    genAmountType genAmount;
    };

<inst-rec>  ->  struct sfInst
    {
    CHAR achInstName[20];
    WORD wInstBagNdx;
    };

<ibag-rec>  ->  struct sfInstBag
    {
    WORD wInstGenNdx;
    WORD wInstModNdx;
    };

<imod-rec> ->  struct sfInstModList
    {
    SFModulator sfModSrcOper;
    SFGenerator sfModDestOper;
    SHORT modAmount;
    SFModulator sfModAmtSrcOper;
    SFTransform sfModTransOper;
    };

<igen-rec>  ->  struct sfInstGenList
    {
```

```
    SFGenerator sfGenOp
    genAmountType genAmount;
    };

<shdr-rec>  ->  struct sfSample
    {
    CHAR achSampleName[20];
    DWORD dwStart;
    DWORD dwEnd;
    DWORD dwStartloop;
    DWORD dwEndloop;
    DWORD dwSampleRate;
    BYTE byOriginalKey;
    CHAR chCorrection;
    WORD wSampleLink;
    SFSampleLink sfSampleType;
    };
```

4.5 SoundFont 2 RIFF File Format Type Definitions

The sfModulator, sfGenerator, and sfTransform types are all enumeration types whose values are defined in subsequent sections.

The genAmountType is a union which allows signed 16 bit, unsigned 16 bit, and two unsigned 8 bit fields:

```
typedef union
    {
    rangesType ranges;
    SHORT shAmount;
    WORD wAmount;
    } genAmountType;

typedef struct
    {
    BYTE byLo;
    BYTE byHi;
    } rangesType;
```

The SFSampleLink is an enumeration type which describes both the type of sample (mono, stereo left, etc.) and the whether the sample is located in RAM or ROM memory:

```
typedef enum
    {
    monoSample = 1,
```

```
rightSample = 2,
leftSample = 4,
linkedSample = 8,
RomMonoSample = 32769,
RomRightSample = 32770,
RomLeftSample = 32772,
RomLinkedSample = 32776
} SFSampleLink;
```

5 The INFO-list Chunk

TheINFO-list chunk in a SoundFont 2 compatible file contains three mandatory and a variety of optional subchunks as defined below. The INFO-list chunk gives basic information about the SoundFont compatible bank contained in the file.

5.1 The ifil Subchunk

The ifil subchunk is a mandatory subchunk identifying the SoundFont specification version level to which the file complies. It is always four bytes in length, and contains data according to the structure:

```
struct sfVersionTag
{
        WORD wMajor;
        WORD wMinor;
};
```

The word wMajor contains the value to the left of the decimal point in the SoundFont specification version, the word wMinor contains the value to the right of the decimal point. For example, version 2.11 would be implied if wMajor=2 and wMinor=11.

These values can be used by applications which read SoundFont compatible files to determine if the format of the file is usable by the program. Within a fixed wMajor, the only changes to the format will be the addition of Generator, Source and Transform enumerators, and additional info subchunks. These are all defined as being ignored if unknown to the program. Consequently, many applications can be designed to be fully upward compatible within a given wMajor. In the case of editors or other programs in which all enumerators should be known, the value of wMinor may be of consequence. Generally the application program will either accept the file as usable (possibly with appropriate transparent translation), reject the file as unusable, or warn the user that there may be uneditable data in the file.

If the ifil subchunk is missing, or its size is not four bytes, the file should be rejected as structurally unsound.

5.2 The isng Subchunk

The isng subchunk is a mandatory subchunk identifying the wavetable sound engine for which the file was optimized. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. The default isng field is the eight bytes representing "EMU8000" as seven ASCII characters followed by a zero byte.

The ASCII should be treated as case-sensitive. In other words "emu8000" is not the same as "EMU8000."

The isng string can be optionally used by chip drivers to vary their synthesis algorithms to emulate the target sound engine.

If the isng subchunk is missing, not terminated in a zero valued byte, or its contents are an unknown sound engine, the field should be ignored and EMU8000 assumed.

5.3 The INAM Subchunk

The INAM subchunk is a mandatory subchunk providing the name of the SoundFont compatible bank. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical inam subchunk would be the fourteen bytes representing "General MIDI" as twelve ASCII characters followed by two zero bytes.

The ASCII should be treated as case-sensitive. In other words "General MIDI" is not the same as "GENERAL MIDI."

The inam string is typically used for the idenitification of banks even if the file names are altered.

If the inam subchunk is missing, or not terminated in a zero valued byte, the field should be ignored and the user supplied with an appropriate error message if the name is queried. If the file is re-written, a valid name should be placed in the INAM field.

5.4 The irom Subchunk

The irom subchunk is an optional subchunk identifying a particular wavetable sound data ROM to which any ROM samples refer. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical irom field would be the six bytes representing "1MGM" as four ASCII characters followed by two zero bytes.

The ASCII should be treated as case-sensitive. In other words "1mgm" is not the same as "1MGM."

The irom string is used by drivers to verify that the ROM data referenced by the file is available to the sound engine.

If the irom subchunk is missi not terminated in a zero valued byte, ' 's contents are an unknown ROM, the field should be ignored and the file assumed to reference no ROM samples. If ROM samples are accessed, any accesses to such intruments should be terminated and not sound. A file should not be written which attempts to access ROM samples without both irom and iver present and valid.

5.5 The iver Subchunk

The iver subchunk is an optional subchunk identifying the particular wavetable sound data ROM revision to which any ROM samples refer. It is always four bytes in length, and contains data according to the structure:

```
struct sfVersionTag
{
        WORD wMajor;
        WORD wMinor;
};
```

The word wMajor contains the value to the left of the decimal point in the ROM version, the word wMinor contains the value to the right of the decimal point. For example, version 1.36 would be implied if wMajor=1 and wMinor=36.

The iver subchunk is used by drivers to verify that the ROM data referenced by the file is located in the exact locations specified by the sound headers.

If the iver subchunk is missing, not four bytes in length, or its contents indicate an unknown or incorrect ROM, the field should be ignored and the file assumed to reference no ROM samples. If ROM samples are accessed, any accesses to such instruments should be terminated and not sound. Note that for ROM samples to function correctly, both iver and irom must be present and valid. A file should not be written which attempts to access ROM samples without both irom and iver present and valid.

5.6 The ICRD Subchunk

The ICRD subchunk is an optional subchunk identifying the creation date of the SoundFont compatible bank. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical ICRD field would be the twelve bytes representing "May 1, 1995" as eleven ASCII characters followed by a zero byte.

Conventionally, the format of the string is "Month Day, Year" where Month is initially capitalized and is the conventional full English spelling of the month, Day is the date in decimal followed by a comma, and Year is the full decimal year. Thus the field should conventionally never be longer than 32 bytes.

The ICRD string is provided for library management purposes.

If the ICRD subchunk is miss not terminated in a zero valued byte, for some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

5.7 The IENG Subchunk

The IENG subchunk is an optional subchunk identifying the names of any sound designers or engineers responsible for the SoundFont compatible bank. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical IENG field would be the twelve bytes representing "Tim Swartz" as ten ASCII characters followed by two zero bytes.

The IENG string is provided for library management purposes.

If the IENG subchunk is missing, not terminated in a zero valued byte, or for some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

5.8 The IPRD Subchunk

The IPRD subchunk is an optional subchunk identifying any specific product for which the SoundFont compatible bank is intended. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical IPRD field would be the eight bytes representing "SBAWE32" as seven ASCII characters followed by a zero byte.

The ASCII should be treated as case-sensitive. In other words "sbawe32" is not the same as "SBAWE32."

The IPRD string is provided for library management purposes.

If the IPRD subchunk is missing, not terminated in a zero valued byte, or for some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

5.9 The ICOP Subchunk

The ICOP subchunk is an optional subchunk containing any copyright assertion string associated with the SoundFont compatible bank. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical ICOP field would be the

40 bytes representing "Copyr t (c) 1995 E-mu Systems, Inc." as 38 ASCII characters followed by two zero bytes.

The ICOP string is provided for intellectual property protection and management purposes.

If the ICOP subchunk is missing, not terminated in a zero valued byte, or for some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

5.10 The ICMT Subchunk

The ICMT subchunk is an optional subchunk containing any comments associated with the SoundFont compatible bank. It contains an ASCII string of 65,536 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical ICMT field would be the 40 bytes representing "This space unintentionally left blank." as 38 ASCII characters followed by two zero bytes.

The ICMT string is provided for any non-scatological uses.

If the ICMT subchunk is missing, not terminated in a zero valued byte, or for some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

5.11 The ISFT Subchunk

The ISFT subchunk is an optional subchunk identifying the SoundFont compatible tools used to create and most recently modify the SoundFont compatible bank. It contains an ASCII string of 256 or fewer bytes including one or two terminators of value zero, so as to make the total byte count even. A typical ISFT field would be the thirty bytes representing "Preditor 2.00a:Preditor 2.00a" as twenty-nine ASCII characters followed by a zero byte.

The ASCII should be treated as case-sensitive. In other words "Preditor" is not the same as "PREDITOR."

Conventionally, the tool name and revision control number are included first for the creating tool and then for the most recent modifying tool. The two strings are separated by a colon. The string should be produced by the creating program with a null modifying tool field (e.g. "Preditor 2.00a:), and each time a tool modifies the bank, it should replace the modifying tool field with its own name and revision control number.

The ISFT string is provided primarily for error tracing purposes.

If the ISFT subchunk is missi not terminated in a zero valued byte, (or some reason incapable of being faithfully copied as an ASCII string, the field should be ignored and if re-written, should not be copied. If the field's contents are not seemingly meaningful but can faithfully reproduced, this should be done.

6 The sdta-list Chunk

The sdta-list chunk in a SoundFont 2 compatible file contains a single optional smpl subchunk which contains all the RAM based sound data associated with the SoundFont compatible bank. The smpl subchunk is of arbitrary length, and contains an even number of bytes.

6.1 Sample Data Format in the smpl Subchunk

The smpl subchunk, if present, contains one or more "samples" of digital audio information in the form of linearly coded sixteen bit, signed, little endian (least significant byte first) words. Each sample is followed by a minimum of forty-six zero valued data points. These zero valued data points are necessary to guarantee that any reasonable upward pitch shift using any reasonable interpolator can loop on zero data at the end of the sound.

6.2 Sample Data Looping Rules

With each sample, one or more loop point pairs may exist. The locations of these points are defined within the pdta-list chunk, but the sample data itself must comply with certain practices in order for the loop to be compatible across multiple platforms.

The loops are defined by "equivalent points" in the sample. This means that there are two samples which are logically equivalent, and a loop occurs when these points are spliced atop one another. In concept, the loop end point is never actually played during looping; instead the loop start point follows the point just prior to the loop end point. Because of the bandlimited nature of digital audio sampling, an artifact free loop will exhibit virtually identical data surrounding the equivalent points.

In actuality, because of the various interpolation algorithms used by wavetable synthesizers, the data surrounding both the loop start and end points may affect the sound of the loop. Hence both the loop start and end points must be surrounded by continuous audio data. For example, even if the sound is programmed to continue to loop throughout the decay, sample data must be provided beyond the loop end point. This data will typically be identical to the data at the start of the loop. A minimum of eight valid data points are required to be present before the loop start and after the loop end.

The eight data points (four on each side) surrounding the two equivalent loop points should also be forced to be identical. By forcing the data to be identical, all interpolation algorithms are guaranteed to properly reproduce an artifact-free loop.

7 The pdta-list Chunk

7.1 The HYDRA Data Structure

The articulation data within a SoundFont 2 compatible file is contained in nine subchunks, named "hydra" after the mythical nine-headed beast. The structure has been designed for interchange purposes; it is not optimized for either run-time synthesis nor for on-the-fly editing. It is reasonable and proper for SoundFont compatible client programs to translate to and from the hydra structure as they read and write SoundFont compatible files.

7.2 The PHDR Subchunk

The PHDR subchunk is a required subchunk listing all presets within the SoundFont compatible file. It is always a multiple of thirty eight bytes in length, and contains a minimum of two records, one record for each preset and one for a terminal record according to the structure:

```
struct sfPresetHeader
{
        CHAR achPresetName[20];
        WORD wPreset;
        WORD wBank;
        WORD wPresetBagNdx;
        DWORD dwLibrary;
        DWORD dwGenre;
        dWORD dwMorphology;
};
```

The ASCII character field achPresetName contains the name of the preset expressed in ASCII, with unused terminal characters filled with zero valued bytes. A unique name should always be assigned to each preset in the SoundFont compatible bank to enable identification. However, if a bank is read containing the erroneous state of presets with identical names, the presets should not be discarded. They should either be preserved as read or preferentially uniquely renamed.

The word wPreset contains the MIDI Preset Number and the word wBank contains the MIDI Bank Number which apply to this preset. Note that the presets are not ordered within the SoundFont compatible bank. Presets should have a unique set of wPreset and wBank numbers. However, if two presets have identical values of both wPreset and wBank, the first occuring preset in the PHDR chunk is the active preset, but any others with the same wBank and wPreset values should be maintained so that they can be renumbered and used at a later time. The special case of a General MIDI percussion bank is handled conventionally by a wBank value of 128. If the value in either field is not a valid MIDI value of zero through 127, or 128 for wBank, the preset cannot be played but should be maintained.

The word wPresetBagNdx is an index to the preset's layer list in the PBAG subchunk. Because the preset layer list is in the same order as the preset header list, the preset bag indicies will be monotonically increasing with increasing preset headers. The size of the PBAG subchunk in bytes will

be equal to four times the ter al preset's wPresetBagNdx plus four. he preset bag indicies are non-monotonic or if the terminal preset's wPresetBagNdx does not match the PBAG subchunk size, the file is structurally defective and should be rejected at load time. All presets except the terminal preset must have at least one layer; any preset with no layers should be ignored.

The doublewords dwLibrary; dwGenre and dwMorphology are reserved for future implementation in a preset library management function and should be preserved as read, and created as zero.

The terminal sfPresetHeader record should never be accessed, and exists only to provide a terminal wPresetBagNdx with which to determine the number of layers in the last preset. All other values are conventionally zero, with the exception of achPresetName, which can optionally be "EOP" indicating end of presets.

If the PHDR subchunk is missing, contains fewer than two records, or its size is not a multiple of 38 bytes, the file should be rejected as structurally unsound.

7.3 The PBAG Subchunk

The PBAG subchunk is a required subchunk listing all preset layers within the SoundFont compatible file. It is always a multiple of four bytes in length, and contains one record for each preset layer plus one record for a terminal layer according to the structure:

```
struct sfPresetBag
{
        WORD wGenNdx;
        WORD wModNdx;
};
```

The first layer in a given preset is located at that preset's wPresetBagNdx. The number of layers in the preset is determined by the difference between the next preset's wPresetBagNdx and the current wPresetBagNdx.

The word wGenNdx is an index to the preset's layer list of generators in the PGEN subchunk, and the wModNdx is an index to its list of modulators in the PMOD subchunk. Because both the generator and modulator lists are in the same order as the preset header and layer lists, these indicies will be monotonically increasing with increasing preset layers. The size of the PMOD subchunk in bytes will be equal to ten times the terminal preset's wModNdx plus ten and the size of the PGEN subchunk in bytes will be equal to four times the terminal preset's wGenNdx plus four. If the generator or modulator indicies are non-monotonic or do not match the size of the respective PGEN or PMOD subchunks, the file is structurally defective and should be rejected at load time.

If a preset has more than one layer, the first layer may be a global layer. A global layer is determined by the fact that the last generator in the list is not an Instrument generator. All generator lists must contain at least one generator with one exception - if a global layer exists for which there are no generators but only modulators. The modulator lists can contain zero or more modulators.

If a layer other than the first layer lacks an Instrument generator as its last generator, that layer should be ignored. A global layer with no modulators and no generators should also be ignored.

If the PBAG subchunk is missing, or its size is not a multiple of four bytes, the file should be rejected as structurally unsound.

7.4 The PMOD Subchunk

The PMOD subchunk is a required subchunk listing all preset layer modulators within the SoundFont compatible file. It is always a multiple of ten bytes in length, and contains zero or more modulators plus a terminal record according to the structure:

```
struct sfModList
{
        SFModulator sfModSrcOper;
        SFGenerator sfModDestOper;
        SHORT modAmount;
        SFModulator sfModAmtSrcOper;
        SFTransform sfModTransOper;
};
```

The preset layer's wModNdx points to the first modulator for that preset layer, and the number of modulators present for a preset layer is determined by the difference between the next higher preset layer's wModNdx and the current preset's wModNdx. A difference of zero indicates there are no modulators in this preset layer.

The sfModSrcOper is a value of one of the SFModulator enumeration type values. Unknown or undefined values are ignored. This value indicates the source of data for the modulator.

The sfModDestOper is a value of one of the SFGenerator enumeration type values. Unknown or undefined values are ignored. This value indicates the destination of the modulator.

The short modAmount is a signed value indicating the degree to which the source modulates the destination. A zero value indicates there is no fixed amount.

The sfModAmtSrcOper is a value of one of the SFModulator enumeration type values. Unknown or undefined values are ignored. This value indicates that a the degree to which the source modulates the destination is to be controlled by the specified modulation source.

The sfModTransOper is a value of one of the SFTransform enumeration type values. Unknown or undefined values are ignored. This value indicates that a transform of the specified type will be applied to the modulation source before application to the modulator.

The terminal record conventionally contains zero in all fields, and is always ignored.

A modulator is defined by its sfModSrcOper, its sfModDestOper, and its sfModSrcAmtOper. All modulators within a layer must have a unique set of these three enumerators. If a second modulator is encountered with the same three enumerators as a previous modulator with the same layer, the first modulator will be ignored.

Modulators in the PMOD subchunk act as additively relative modulators with respect to those in the IMOD subchunk. In other words, a PMOD modulator can increase or decrease the amount of an IMOD modulator.

If the PMOD subchunk is missing, or its size is not a multiple of ten bytes, the file should be rejected as structurally unsound.

7.5 The PGEN Subchunk

The PGEN chunk is a required chunk containing a list of preset layer generators for each preset layer within the SoundFont compatible file. It is always a multiple of four bytes in length, and contains one or more generators for each preset layer (except a global layer containing only modulators) plus a terminal record according to the structure:

```
struct sfGenList
{
        SFGenerator sfGenOper;
        genAmountType genAmount;
};
```

where the types are defined:

```
typedef union
{
        rangesType ranges;
        SHORT shAmount;
        WORD wAmount;
} genAmountType;

typedef struct
{
        BYTE byLo;
        BYTE byHi;
} rangesType;
```

The sfGenOper is a value of one of the SFGenerator enumeration type values. Unknown or undefined values are ignored. This value indicates the type of generator being indicated.

The genAmount is the value be assigned to the specified generator ote that this can be of three formats. Certain generators specify a range of MIDI key numbers of MIDI velocities, with a minimum and maximum value. Other generators specify an unsigned WORD value. Most generators, however, specify a signed 16 bit SHORT value.

The preset layer's wGenNdx points to the first generator for that preset layer. Unless the layer is a global layer, the last generator in the list is an "Instrument" generator, whose value is a pointer to the instrument associated with that layer. If a "key range" generator exists for the preset layer, it is always the first generator in the list for that preset layer. If a "velocity range" generator exists for the preset layer, it will only be preceded by a key range generator. If any generators follow an Instrument generator, they will be ignored.

A generator is defined by its sfGenOper. All generators within a layer must have a unique sfGenOper enumerator. If a second generator is encountered with the same sfGenOper enumerator as a previous generator with the same layer, the first generator will be ignored.

Generators in the PGEN subchunk act as additively relative to generators in the IGEN subchunk. In other words, PGEN generators increase or decrease the value of an IGEN generator.

If the PGEN subchunk is missing, or its size is not a multiple of four bytes, the file should be rejected as structurally unsound. If a key range generator is present and not the first generator, it should be ignored. If a velocity range generator is present, and is preceded by a generator other than a key range generator, it should be ignored. If a non-global list does not end in an instrument generator, layer should be ignored. If the instrument generator value is equal to or greater than the terminal instrument, the file should be rejected as structurally unsound.

7.6 The INST Subchunk

The inst subchunk is a required subchunk listing all instruments within the SoundFont compatible file. It is always a multiple of twenty two bytes in length, and contains a minimum of two records, one record for each instrument and one for a terminal record according to the structure:

```
struct sfInst
{
        CHAR achInstName[20];
        WORD wInstBagNdx;
};
```

The ASCII character field achInstName contains the name of the instrument expressed in ASCII, with unused terminal characters filled with zero valued bytes. A unique name should always be assigned to each instrument in the SoundFont compatible bank to enable identification. However, if a bank is read containing the erroneous state of instruments with identical names, the instruments should not be discarded. They should either be preserved as read or preferentially uniquely renamed.

The word wInstBagNdx is a dex to the instrument's split list in the AG subchunk. Because the instrument split list is in the same order as the instrument list, the instrument bag indicies will be monotonically increasing with increasing instruments. The size of the IBAG subchunk in bytes will be equal to four times the terminal instrument's wInstBagNdx plus four. If the instrument bag indicies are non-monotonic or if the terminal instrument's wInstBagNdx does not match the IBAG subchunk size, the file is structurally defective and should be rejected at load time. All instruments except the terminal instrument must have at least one split, any preset with no splits should be ignored.

The terminal sfInst record should never be accessed, and exists only to provide a terminal wInstBagNdx with which to determine the number of splits in the last instrument. All other values are conventionally zero, with the exception of achInstName, which can optionally be "EOI" indicating end of instruments.

If the INST subchunk is missing, contains fewer than two records, or its size is not a multiple of 22 bytes, the file should be rejected as structurally unsound. All instruments present in the inst subchunk are typically referenced by a preset layer, however a file containing any "orphaned" instruments need not be rejected. SoundFont compatible applications can optionally ignore or filter out these orphaned instruments based on user preference.

7.7 The IBAG Subchunk

The IBAG subchunk is a required subchunk listing all instrument splits within the SoundFont compatible file. It is always a multiple of four bytes in length, and contains one record for each instrument split plus one record for a terminal layer according to the structure:

```
struct sfInstBag
{
        WORD wInstGenNdx;
        WORD wInstModNdx;
};
```

The first split in a given instrument is located at that instrument's wInstBagNdx. The number of splits in the instrument is determined by the difference between the next instrument's wInstBagNdx and the current wInstBagNdx.

The word wInstGenNdx is an index to the instrument split's list of generators in the IGEN subchunk, and the wInstModNdx is an index to its list of modulators in the IMOD subchunk. Because both the generator and modulator lists are in the same order as the instrument and split lists, these indicies will be monotonically increasing with increasing splits. The size of the IMOD subchunk in bytes will be equal to ten times the terminal instrument's wModNdx plus ten and the size of the IGEN subchunk in bytes will be equal to four times the terminal instrument's wGenNdx plus four. If the generator or modulator indicies are non-monotonic or do not match the size of the respective IGEN or IMOD subchunks, the file is structurally defective and should be rejected at load time.

If an instrument has more than one split, the first split may be a global split. A global split is determined by the fact that the last generator in the list is not an sampleID generator. All generator lists

must contain at least one gen .tor with one exception - if a global sp xists for which there are no generators but only modulators. The modulator lists can contain zero or more modulators.

If a split other than the first split lacks an sampleID generator as its last generator, that split should be ignored. A global split with no modulators and no generators should also be ignored.

If the IBAG subchunk is missing, or its size is not a multiple of four bytes, the file should be rejected as structurally unsound.

7.8 The IMOD Subchunk

The IMOD subchunk is a required subchunk listing all instrument split modulators within the SoundFont compatible file. It is always a multiple of ten bytes in length, and contains zero or more modulators plus a terminal record according to the structure:

```
struct sfModList
{
        SFModulator sfModSrcOper;
        SFGenerator sfModDestOper;
        SHORT modAmount;
        SFModulator sfModAmtSrcOper;
        SFTransform sfModTransOper;
};
```

The split's wInstModNdx points to the first modulator for that split, and the number of modulators present for a split is determined by the difference between the next higher split's wInstModNdx and the current split's wModNdx. A difference of zero indicates there are no modulators in this split.

The sfModSrcOper is a value of one of the SFModulator enumeration type values. Unknown or undefined values are ignored. This value indicates the source of data for the modulator.

The sfModDestOper is a value of one of the SFGenerator enumeration type values. Unknown or undefined values are ignored. This value indicates the destination of the modulator.

The short modAmount is a signed value indicating the degree to which the source modulates the destination. A zero value indicates there is no fixed amount.

The sfModAmtSrcOper is a value of one of the SFModulator enumeration type values. Unknown or undefined values are ignored. This value indicates that a the degree to which the source modulates the destination is to be controlled by the specified modulation source.

The sfModTransOper is a value of one of the SFTransform enumeration type values. Unknown or undefined values are ignored. This value indicates that a transform of the specified type will be applied to the modulation source before application to the modulator.

If the IGEN subchunk is missing, or its size is not a multiple of four bytes, the file should be rejected as structurally unsound. If a key range generator is present and not the first generator, it should be ignored. If a velocity range generator is present, and is preceded by a generator other than a key range generator, it should be ignored. If a non-global list does not end in a sampleID generator, the split should be ignored. If the sampleID generator value is equal to or greater than the terminal sampleID, the file should be rejected as structurally unsound.

7.10 The SHDR Subchunk

The SHDR chunk is a required subchunk listing all samples within the smpl subchunk and any referenced ROM samples. It is always a multiple of forty six bytes in length, and contains one record for each sample plus a terminal record according to the structure:

```
struct sfSample
{
        CHAR achSampleName[20];
        DWORD dwStart;
        DWORD dwEnd;
        DWORD dwStartloop;
        DWORD dwEndloop;
        DWORD dwSampleRate;
        BYTE byOriginalPitch;
        CHAR chPitchCorrection;
        WORD wSampleLink;
        SFSampleLink sfSampleType;
};
```

The ASCII character field achSampleName contains the name of the sample expressed in ASCII, with unused terminal characters filled with zero valued bytes. A unique name should always be assigned to each sample in the SoundFont compatible bank to enable identification. However, if a bank is read containing the erroneous state of samples with identical names, the samples should not be discarded. They should either be preserved as read or preferentially uniquely renamed.

The doubleword dwStart contains the index, in samples, from the beginning of the sample data field to the first data point of this sample.

The doubleword dwEnd contains the index, in samples, from the beginning of the sample data field to the first of the set of 46 zero valued data points following this sample.

The doubleword dwStartloop contains the index, in samples, from the beginning of the sample data field to the first datapoint in the loop of this sample.

The doubleword dwEndloop contains the index, in samples, from the beginning of the sample data field to the first datapoint following the loop of this sample. Note that this is the data point "equivalent to"

the first loop datapoint, and t to produce portable artifact free loops e sixteen proximal datapoints surrounding both the Startloop and Endloop points should be identical.

The values of dwStart, dwEnd, dwStartloop, and dwEndloop must all be within the range of the sample data field included in the SoundFont compatible bank or referenced in the sound ROM. Also, to allow a variety of hardware platforms to be able to reproduce the data, the samples have a minimum length of 48 data points, a minimum loop size of 32 data points, and a minimum of 8 valid points prior to dwStartloop and after dwEndloop. Thus dwStart must be less than dwStartloop-7, dwStartloop must be less than dwEndloop-31, and dwEndloop must be less than dwEnd-7. If these constraints are not met, the sound may optionally not be played if the hardware cannot support artifact-free playback for the parameters given.

The doubleword dwSampleRate contains the sample rate, in Hertz, at which this sample was acquired or to which it was most recently converted. Values of greater than 50000 or less than 400 may not be reproducable by some hardware platforms and should be avoided. A value of zero is illegal. If an illegal or impractical value is encountered, the nearest practical value should be used.

The byte byOriginalPitch contains the MIDI key number of the recorded pitch of the sample. For example, a recording of an instrument playing middle C (261.62 Hz) should receive a value of 60. This value is used as the default "root key" for the sample, so that in the example, a MIDI key-on command for note number 60 would reproduce the sound at its original pitch. For unpitched sounds, a conventional value of 255 should be used. Values between 128 and 254 are illegal. Whenever an illegal value or a value of 255 is encountered, the value 60 should be used.

The character chPitchCorrection contains a pitch correction in cents which should be applied to the sample on playback. The purpose of this field is to compensate for any pitch errors during the sample recording process. The correction value is that of the correction to be applied. For example, if the sound is 4 cents sharp, a correction bringing it 4 cents flat is required, thus the value should be -4.

The value in sfSampleType is an enumeration with eight defined values: monoSample = 1, rightSample = 2, leftSample = 4, linkedSample = 8, RomMonoSample = 32769, RomRightSample = 32770, RomLeftSample = 32772, and RomLinkedSample = 32776. It can be seen that this is encoded such that bit 15 of the 16 bit value is set if the sample is in ROM, and reset if it is included in the SoundFont compatible bank. The four LS bits of the word are then exclusively set indicating mono, left, right, or linked.

If the sound is flagged as a ROM sample and no valid IROM subchunk is included, the file is structurally defective and should be rejected at load time.

If sfSampleType indicates a mono sample, then wSampleLink is undefined and its value should be conventionally zero, but will be ignored regardless of value. If sfSampleType indicates a left or right sample, then wSampleLink is the sample header index of the associated right or left stereo sample respectively. Both samples should be played together, with their pans forced to the appropriate direction. The linked sample type is not currently fully defined in the SoundFont 2 specification, but will ultimately support a circularly linked list of samples using wSampleLink.

The terminal sample record is never referenced, and is conventionally entirely zero with the exception of achSampleName, which can optionally be "EOS" indicating end of samples. All samples present in the smpl subchunk are typically referenced by an instrument, however a file containing any "orphaned" samples need not be rejected. SoundFont compatible applications can optionally ignore or filter out these orphaned samples according to user preference.

If the SHDR subchunk is missing, or its is size is not a multiple of 46 bytes the file should be rejected as structurally unsound.

APPENDIX II

S.1.2 Generator Enumerators Defined

The following is an exhaustive list of SoundFont 2.00 generators and their strict definitions:

0	startAddrsOffset	The offset, in samples, beyond the Start sample header parameter to the first sample to be played for this instrument. For example, if Start were 7 and startAddrOffset were 2, the first sample played would be sample 9.

1	endAddrsOffset	The offset, in samples, beyond the E sample header parameter to the last sample to be played for this instrument. For example, if End were 17 and endAddrOffset were -2, the last sample played would be sample 15.
2	startloopAddrsOffset	The offset, in samples, beyond the Startloop sample header parameter to the first sample to be repeated in the loop for this instrument. For example, if Startloop were 10 and startloopAddrOffset were -1, the first repeated loop sample would be sample 9.
3	endloopAddrsOffset	The offset, in samples, beyond the Endloop sample header parameter to the sample considered equivalent to the Startloop sample for the loop for this instrument. For example, if Endloop were 15 and endloopAddrOffset were 2, sample 17 would be considered equivalent to the Startloop sample, and hence sample 16 would effectively precede Startloop during looping.
4	startAddrsCoarseOffset	The offset, in 32768 sample increments beyond the Start sample header parameter and the first sample to be played in this instrument. This parameter is added to the startAddrsOffset parameter. For example, if Start were 5, startAddrOffset were 3 and startAddrCoarseOffset were 2, the first sample played would be sample 65544.
5	modLfoToPitch	This is the degree, in cents, to which a full scale excursion of the Modulation LFO will influence pitch. A positive value indicates a positive LFO excursion increases pitch; a negative value indicates a positive excursion decreases pitch. Pitch is always modified logarithmically, that is the deviation is in cent, semitones, and octaves rather than in Hz. For example, a value of 100 indicates that the pitch will first rise 1 semitone, then fall one semitone.
6	vibLfoToPitch	This is the degree, in cents, to which a full scale excursion of the Vibrato LFO will influence pitch. A positive value indicates a positive LFO excursion increases pitch; a negative value indicates a positive excursion decreases pitch. Pitch is always modified logarithmically, that is the deviation is in cent, semitones, and octaves rather than in Hz. For example, a value of 100 indicates that the pitch will first rise 1 semitone, then fall one semitone.
7	modEnvToPitch	This is the degree, in cents, to which a full scale excursion of the Modulation Envelope will influence pitch. A positive value indicates an increase in pitch; a negative value indicates a decrease in pitch. Pitch is always modified logarithmically, that is the deviation is in cent, semitones, and octaves rather than in Hz. For example, a value

		of 100 indicates that the pitch will ri semitone at the envelope peak.
8	initialFilterFc	This is the cutoff and resonant frequency of the lowpass filter in absolute cent units. The lowpass filter is defined as a second order resonant pole pair whose pole frequency in Hz is defined by the Initial Filter Cutoff parameter. When the cutoff frequency exceeds 20kHz and the Q (resonance) of the filter is zero, the filter does not affect the signal.
9	initialFilterQ	This is the height above DC gain in centibels which the filter resonance exhibits at the cutoff frequency. A value of zero or less indicates the filter is not resonant; the gain at the cutoff frequency (pole angle) may be less than zero when zero is specified. The filter gain at DC is also affected by this parameter such that the gain at DC is reduced by half the specified gain. For example, for a value of 100, the filter gain at DC would be 5 dB below unity gain, and the height of the resonant peak would be 10 dB above the DC gain, or 5 dB above unity gain. Note also that if initialFilterQ is set to zero or less, then the filter response is flat and unity gain if the cutoff frequency exceeds 20 kHz.
10	modLfoToFilterFc	This is the degree, in cents, to which a full scale excursion of the Modulation LFO will influence filter cutoff frequency. A positive number indicates a positive LFO excursion increases cutoff frequency; a negative number indicates a positive excursion decreases cutoff frequency. Filter cutoff frequency is always modified logarithmically, that is the deviation is in cent, semitones, and octaves rather than in Hz. For example, a value of 1200 indicates that the cutoff frequency will first rise 1 octave, then fall one octave.
11	modEnvToFilterFc	This is the degree, in cents, to which a full scale excursion of the Modulation Envelope will influence filter cutoff. A positive number indicates an increase in cutoff frequency; a negative number indicates a decrease in filter cutoff. Filter cutoff is always modified logarithmically, that is the deviation is in cent, semitones, and octaves rather than in Hz. For example, a value of 1000 indicates that the cutoff frequency will rise one octave at the envelope attack peak.
12	endAddrsCoarseOffset	The offset, in 32768 sample increments beyond the End sample header parameter and the last sample to be played in this instrument. This parameter is added to the endAddrsOffset parameter. For example, if End were 65536, startAddrOffset were -3 and startAddrCoarseOffset were -1, the last sample played would be sample 32765.

13 modLfoToVolume	This is the degree, in centibels, to wh a full scale excursion of the Modulation LFO will influence volume. A positive number indicates a positive LFO excursion increases volume; a negative number indicates a positive excursion decreases volume. Volume is always modified logarithmically, that is the deviation is in decibels rather than in linear amplitude. For example, a value of 100 indicates that the volume will first rise ten dB, then fall ten dB.
14 unused1	Unused, reserved. Should be ignored if encountered.
15 chorusEffectsSend	This is the degree, in 0.1% units, to which the audio output of the note is sent to the chorus effects processor. A value of 0% or less indicates no signal is sent from this note; a value of 100% or more indicates the note is sent at full level. Note that this parameter has no effect on the amount of this signal sent to the "dry" or unprocessed portion of the output. For example, a value of 250 indicates that the signal is sent at 25% of full level (attenuation of 12 dB from full level) to the chorus effects processor.
16 reverbEffectsSend	This is the degree, in 0.1% units, to which the audio output of the note is sent to the reverb effects processor. A value of 0% or less indicates no signal is sent from this note; a value of 100% or more indicates the note is sent at full level. Note that this parameter has no effect on the amount of this signal sent to the "dry" or unprocessed portion of the output. For example, a value of 250 indicates that the signal is sent at 25% of full level (attenuation of 12 dB from full level) to the reverb effects processor.
17 pan	This is the degree, in 0.1% units, to which the "dry" audio output of the note is positioned to the left or right output. A value of -50% or less indicates the signal is sent entirely to the left output and not sent to the right output; a value of +50% or more indicates the note is sent entirely to the right and not sent to the left. A value of zero places the signal centered between left and right. For example, a value of -250 indicates that the signal is sent at 75% of full level to the left output and 25% of full level to the right output.
18 unused2	Unused, reserved. Should be ignored if encountered.
19 unused3	Unused, reserved. Should be ignored if encountered.
20 unused4	Unused, reserved. Should be ignored if encountered.
21 delayModLFO	This is the delay time, in absolute timecents, from key on until the Modulation LFO begins its upward ramp from zero value. A value of 0 indicates a 1 second delay. A negative value indicates a delay less

than one second; a positive value a d / longer than one second. The most negative number (-32768) conventionally indicates no delay. For example, a delay of 10 msec would be 1200log2(.01) = -7973.

22 freqModLFO — This is the frequency, in absolute cents, of the Modulation LFO's triangular period. A value of zero indicates a frequency of 8.176 Hz. A negative value indicates a frequency less than 8.176 Hz; a positive value a frequency greater than 8.176 Hz. For example, a frequency of 10 mHz would be 1200log2(.01/8.176) = -11610.

23 delayVibLFO — This is the delay time, in absolute timecents, from key on until the Vibrato LFO begins its upward ramp from zero value. A value of 0 indicates a 1 second delay. A negative value indicates a delay less than one second; a positive value a delay longer than one second. The most negative number (-32768) conventionally indicates no delay. For example, a delay of 10 msec would be 1200log2(.01) = -7973.

24 freqVibLFO — This is the frequency, in absolute cents, of the Vibrato LFO's triangular period. A value of zero indicates a frequency of 8.176 Hz. A negative value indicates a frequency less than 8.176 Hz; a positive value a frequency greater than 8.176 Hz. For example, a frequency of 10 mHz would be 1200log2(.01/8.176) = -11610.

25 delayModEnv — This is the delay time, in absolute timecents, between key on and the start of the attack phase of the Modulation envelope. A value of 0 indicates a 1 second delay. A negative value indicates a delay less than one second; a positive value a delay longer than one second. The most negative number (-32768) conventionally indicates no delay. For example, a delay of 10 msec would be 1200log2(.01) = -7973.

26 attackModEnv — This is the time, in absolute timecents, from the end of the Modulation Envelope Delay Time until the point at which the Modulation Envelope value reaches its peak. Note that the attack is "convex"; the curve is nominally such that when applied to a decibel or semitone parameter, the result is linear in amplitude or Hz respectively. A value of 0 indicates a 1 second attack time. A negative value indicates a time less than one second; a positive value a time longer than one second. The most negative number (-32768) conventionally indicates instantaneous attack. For example, an attack time of 10 msec would be 1200log2(.01) = -7973.

27 holdModEnv — This is the time, in absolute timecents, from the end of the attack phase to the entry into decay phase, during which the envelope value is held at its peak. A value of 0 indicates a 1 second hold time. A negative value indicates a time less than one second; a positive value a time longer than one second. The most negative number (-32768)

	conventionally indicates no hold pha For example, a hold time of 10 msec would be 1200log2(.01) = -7973.
28 decayModEnv	This is the time, in absolute timecents, for a 100% change in the Modulation Envelope value during decay phase. For the Modulation Envelope, the decay phase linearly ramps toward the sustain level. If the sustain level were zero, the Modulation Envelope Decay Time would be the time spent in decay phase. A value of 0 indicates a 1 second decay time for a zero sustain level. A negative value indicates a time less than one second; a positive value a time longer than one second. For example, a decay time of 10 msec would be 1200log2(.01) = -7973.
29 sustainModEnv	This is the decrease in level, expressed in 0.1% units, over which the Modulation Envelope value ramps during the decay phase. For the Modulation Envelope, the sustain level is best expressed in percent of full scale. For congruity with the volume envelope, the sustain level is expressed as a decrease from full scale. A value of 0 indicates the sustain level is full level; this implies a zero duration of decay phase regardless of decay time. A positive value indicates a decay to the corresponding level. Values less than zero are to be interpreted as zero; values above 1000 are to be interpreted as 1000. For example, a sustain level which coresponds to an absolute value 40% of peak would be 600.
30 releaseModEnv	This is the time, in absolute timecents, for a 100% change in the Modulation Envelope value during release phase. For the Modulation Envelope, the release phase linearly ramps toward zero from the current level. If the current level were full scale, the Modulation Envelope Release Time would be the time spent in release phase until zero value were reached. A value of 0 indicates a 1 second decay time for a release from full level. A negative value indicates a time less than one second; a positive value a time longer than one second. For example, a release time of 10 msec would be 1200log2(.01) = -7973.
31 keynumToModEnvHold	This is the degree, in timecent per keynumber units, to which the hold time of the Modulation Envelope is decreased by increasing MIDI key number. The hold time at key number 60 is always unchanged. The unit scaling is such that a value of 100 provides a hold time which tracks the keyboard, that is an upward octave causes the hold time to halve. For example, if the Modulation Envelope Hold Time were -7973 = 10 msec and the Key Number to Mod Env Hold were 50, when a key number 36 was played, the hold time would be 20 msec.
32 keynumToModEnvDecay	This is the degree, in timecent per keynumber units, to which the hold time of the Modulation Envelope is decreased by increasing MIDI key

number. The hold time at key numb '0 is always unchanged. The unit scaling is such that a value of 100 provides a hold time which tracks the keyboard, that is an upward octave causes the hold time to halve. For example, if the Modulation Envelope Hold Time were -7973 = 10 msec and the Key Number to Mod Env Hold were 50, when a key number 36 was played, the hold time would be 20 msec.

33 delayVolEnv — This is the delay time, in absolute timecents, between key on and the start of the attack phase of the Volume envelope. A value of 0 indicates a 1 second delay. A negative value indicates a delay less than one second; a positive value a delay longer than one second. The most negative number (-32768) conventionally indicates no delay. For example, a delay of 10 msec would be 1200log2(.01) = -7973.

34 attackVolEnv — This is the time, in absolute timecents, from the end of the Volume Envelope Delay Time until the point at which the Volume Envelope value reaches its peak. Note that the attack is "convex"; the curve is nominally such that when applied to the decibel volume parameter, the result is linear in amplitude. A value of 0 indicates a 1 second attack time. A negative value indicates a time less than one second; a positive value a time longer than one second. The most negative number (-32768) conventionally indicates instantaneous attack. For example, an attack time of 10 msec would be 1200log2(.01) = -7973.

35 holdVolEnv — This is the time, in absolute timecents, from the end of the attack phase to the entry into decay phase, during which the Volume envelope value is held at its peak. A value of 0 indicates a 1 second hold time. A negative value indicates a time less than one second; a positive value a time longer than one second. The most negative number (-32768) conventionally indicates no hold phase. For example, a hold time of 10 msec would be 1200log2(.01) = -7973.

36 decayVolEnv — This is the time, in absolute timecents, for a 100% change in the Volume Envelope value during decay phase. For the Volume Envelope, the decay phase linearly ramps toward the sustain level, causing a constant dB change for each time unit. If the sustain level were -100dB, the Volume Envelope Decay Time would be the time spent in decay phase. A value of 0 indicates a 1 second decay time for a zero sustain level. A negative value indicates a time less than one second; a positive value a time longer than one second. For example, a decay time of 10 msec would be 1200log2(.01) = -7973.

37 sustainVolEnv — This is the decrease in level, expressed in centibels, over which the Volume Envelope value ramps during the decay phase. For the Volume Envelope, the sustain level is best expressed in cB of attenuation from full scale. A value of 0 indicates the sustain level is

	full level; this implies a zero duration decay phase regardless of decay time. A positive value indicates a decay to the corresponding level. Values less than zero are to be interpreted as zero; conventionally 1000 indicates full attenuation. For example, a sustain level which coresponds to an absolute value 12dB below of peak would be 120.
38 releaseVolEnv	This is the time, in absolute timecents, for a 100% change in the Volume Envelope value during release phase. For the Volume Envelope, the release phase linearly ramps toward zero from the current level, causing a constant dB change for each time unit. If the current level were full scale, the Volume Envelope Release Time would be the time spent in release phase until -100dB attenuation were reached. A value of 0 indicates a 1 second decay time for a release from full level. A negative value indicates a time less than one second; a positive value a time longer than one second. For example, a release time of 10 msec would be 1200log2(.01) = -7973.
39 keynumToVolEnvHold	This is the degree, in timecent per keynumber units, to which the hold time of the Volume Envelope is decreased by increasing MIDI key number. The hold time at key number 60 is always unchanged. The unit scaling is such that a value of 100 provides a hold time which tracks the keyboard, that is an upward octave causes the hold time to halve. For example, if the Volume Envelope Hold Time were -7973 = 10 msec and the Key Number to Vol Env Hold were 50, when a key number 36 was played, the hold time would be 20 msec.
40 keynumToVolEnvDecay	This is the degree, in timecent per keynumber units, to which the hold time of the Volume Envelope is decreased by increasing MIDI key number. The hold time at key number 60 is always unchanged. The unit scaling is such that a value of 100 provides a hold time which tracks the keyboard, that is an upward octave causes the hold time to halve. For example, if the Volume Envelope Hold Time were -7973 = 10 msec and the Key Number to Vol Env Hold were 50, when a key number 36 was played, the hold time would be 20 msec.
41 instrument	This is the index into the INST subchunk providing the instrument to be used for the current layer. A value of zero indicates the first instrument in the list. The value should never exceed the size of the instrument list. The instrument enumerator is the terminal generator for PGEN layers. As such, it should only appear in the PGEN subchunk, and it must appear as the last generator enumerator in all but the global layer.
42 reserved1	Unused, reserved. Should be ignored if encountered.

43 keyRange	This is the minimum and maximum MIDI key number values for which this preset, layer, instrument or split is active. The LS byte indicates the highest and the MS byte the lowest valid key . The keyRange enumerator is optional, but when it does appear, it must be the first generator in the preset, layer, instrument or split.
44 velRange	This is the minimum and maximum MIDI velocity values for which this preset, layer, instrument or split is active. The LS byte indicates the highest and the MS byte the lowest valid velocity . The velRange enumerator is optional, but when it does appear, it must be preceded only by keyRange in the preset, layer, instrument or split.
45 startloopAddrsCoarseOffset	The offset, in 32768 sample increments beyond the Startloop sample header parameter and the first sample to be repeated in this instrument's loop. This parameter is added to the startloopAddrsOffset parameter. For example, if Startloop were 5, startloopAddrOffset were 3 and startAddrCoarseOffset were 2, the first sample in the loop would be sample 65544.
46 keynum	This enumerator forces the MIDI key number to effectively be interpreted as the value given. Valid values are from 0 to 127.
47 velocity	This enumerator forces the MIDI velocity to effectively be interpreted as the value given. Valid values are from 0 to 127.
48 initialAttenuation	This is the attenuation, in centibels, by which a note is attenuated below full scale. A value of zero indicates no attenuation; the note will be played at full scale. For example, a value of 60 indicates the note will be played at 6 dB below full scale for the note.
49 reserved2	Unused, reserved. Should be ignored if encountered.
50 endloopAddrsCoarseOffset	The offset, in 32768 sample increments beyond the Endloop sample header parameter parameter to the sample considered equivalent to the Startloop sample for the loop for this instrument. This parameter is added to the endloopAddrsOffset parameter. For example, if Endloop were 5, endloopAddrOffset were 3 and endAddrCoarseOffset were 2, sample 65544 would be considered equivalent to the Startloop sample, and hence sample 65543 would effectively precede Startloop during looping.
51 coarseTune	This is a pitch offset, in semitones, which should be applied to the note. A positive value indicates the sound is reproduced at a higher pitch; a negative value indicates a lower pitch. For example, a Coarse Tune value of -4 would cause the sound to be reproduced four semitones flat.

52 fineTune	This is a pitch offset, in cents, which should be applied to the note. It is additive with coarseTune. A positive value indicates the sound is reproduced at a higher pitch; a negative value indicates a lower pitch. . For example, a Fine Tuning value of -5 would cause the sound to be reproduced five cents flat.
53 sampleID	This is the index into the SHDR subchunk providing the sample to be used for the current split. A value of zero indicates the first sample in the list. The value should never exceed the size of the sample list. The sampleID enumerator is the terminal generator for IGEN splits. As such, it should only appear in the IGEN subchunk, and it must appear as the last generator enumerator in all but the global split.
54 sampleModes	This enumerator indicates a value which gives a variety of Boolean flags describing the sample for the current instrument split. The sampleModes should only appear in the IGEN subchunk, and should not appear in the global split. The two LS bits of the value indicate the type of loop in the sample: 0 indicates a sound reproduced with no loop, 1 indicates a sound which loops continuously, 2 redundantly indicates no loop, and 3 indicates a sound which loops for the duration of key depression then proceeds to play the remainder of the sample. The MS bit (bit 15) of the value indicates that this sample is found in the ROM memory of the sound engine.
55 reserved3	Unused, reserved. Should be ignored if encountered.
56 scaleTuning	This parameter represents the degree to which MIDI key number influences pitch. A value of zero indicates that MIDI key number has no effect on pitch; a value of 100 represents the usual tempered semitone scale.
57 exclusiveClass	This parameter provides the capability for a key depression in a given instrument to terminate the playback of other instruments. This is particularly useful for percussive instruments such as a hihat cymbal. An exclusive class value of zero indicates no exclusive class; no special action is taken. Any other value indicates that when this note is initiated, any other sounding note with the same exclusive class value should be rapidly terminated.
58 overridingRootKey	This parameter represents the MIDI key number at which the sample is to be played back at its original sample rate. If not present, or if present with a value of -1, then the sample header parameter Original Key is used in its place. If it is present in the range 0-127, then the indicated key number will cause the sample to be played back at its sample header Sample Rate. For example, if the sample were a

recording of a piano middle C (Orig Key = 60) at a sample rate of 22.050 kHz, and Root Key were set to 69, then playing MIDI key number 69 (A above middle C) would cause a piano note of pitch middle C to be heard.

59 unused5 — Unused, reserved. Should be ignored if encountered.

60 endOper — Unused, reserved. Should be ignored if encountered. Unique name provides value to end of defined list.

8.1.3 Generator Summary

The following tables give the ranges and default values for all SoundFont 2.00 defined generators.

#	Name	Unit	Abs Zero	Min Useful		Max Useful		Default Value
0	startAddrsOffset	smpls	0	0	None	*	*	0 None
1	endAddrsOffset	smpls	0	*	*	0	None	0 None
2	startloopAddrsOffset	smpls	0	*	*	*	*	0 None
3	endloopAddrsOffset	smpls	0	*	*	*	*	0 None
4	startAddrsCoarseOffset	32k smpls	0	0	None	*	*	0 None
5	modLfoToPitch	cent fs	0	-12000	-10 oct	12000	10 oct	0 None
6	vibLfoToPitch	cent fs	0	-12000	-10 oct	12000	10 oct	0 None
7	modEnvToPitch	cent fs	0	-12000	-10 oct	12000	10 oct	0 None
8	initialFilterFc	cent	8.176 Hz	1500	20 Hz	13500	20 kHz	13500 Open
9	initialFilterQ	cB	0	0	None	960	96 dB	0 None
10	modLfoToFilterFc	cent fs	0	-12000	-10 oct	12000	10 oct	0 None
11	modEnvToFilterFc	cent fs	0	-12000	-10 oct	12000	10 oct	0 None
12	endAddrsCoarseOffset	32k smpls	0	*	*	0	None	0 None
13	modLfoToVolume	cbB fs	0	-960	-96 dB	960	96 dB	0 None
15	chorusEffectsSend	0.1%	0	0	None	1000	100%	0 None
16	reverbEffectsSend	0.1%	0	0	None	1000	100%	0 None
17	pan	0.1%	Center	-500	Left	+500	Right	0 Center
21	delayModLFO	timecent	1 sec	-12000	1 msec	5000	20 sec	-12000 <1 msec
22	freqModLFO	cent	8.176 Hz	-16000	1 mHz	4500	100 Hz	0 8.176 Hz
23	delayVibLFO	timecent	1 sec	-12000	1 msec	5000	20 sec	-12000 <1 msec
24	freqVibLFO	cent	8.176 Hz	-16000	1 mHz	4500	100 Hz	0 8.176 Hz
25	delayModEnv	timecent	1 sec	-12000	1 msec	5000	20 sec	-12000 <1 msec
26	attackModEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000 <1 msec
27	holdModEnv	timecent	1 sec	-12000	1 msec	5000	20 sec	-12000 <1 msec
28	decayModEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000 <1 msec
29	sustainModEnv	-0.1%	attk peak	0	100%	1000	0%	0 attk pk
30	releaseModEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000 <1 msec
31	keynumToModEnvHold	tcent/key	0	-1200	-oct/ky	1200	oct/ky	0 None
32	keynumToModEnvDecay	tcent/key	0	-1200	-oct/ky	1200	oct/ky	0 None
33	delayVolEnv	timecent	1 sec	-12000	1 msec	5000	20 sec	-12000 <1 msec
34	attackVolEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000 <1 msec

35	holdVolEnv	timecent	1 sec	-12000	1 msec	8000	20 sec	-12000	<1 msec
36	decayVolEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000	<1 msec
37	sustainVolEnv	cB atn	atk peak	0	0 dB	1440	144dB	0	atk pk
38	releaseVolEnv	timecent	1 sec	-12000	1 msec	8000	100sec	-12000	<1 msec
39	keynumToVolEnvHold	tcent/key	0	-1200	-oct/ky	1200	oct/ky	0	None
40	keynumToVolEnvDecay	tcent/key	0	-1200	-oct/ky	1200	oct/ky	0	None
43	keyRange	MIDI ky#	key# 0	0	lo key	127	hi key	0-127	full kbd
44	velRange	MIDI vel	0	0	min vel	127	mx vel	0-127	all vels
45	startloopAddrsCoarseOffset	smpls	0	*	*	*	*	0	None
46	keynum	MIDI ky#	key# 0	0	lo key	127	hi key	-1	None
47	velocity	MIDI vel	0	1	min vel	127	mx vel	-1	None
48	initialAttenuation	cB	0	0	0 dB	1440	144dB	0	None
50	endloopAddrsCoarseOffset	smpls	0	*	*	*	*	0	None
51	coarseTune	semitone	0	-120	-10 oct	120	10 oct	0	None
52	fineTune	cent	0	-99	-99cent	99	99cent	0	None
54	sampleModes	Bit Flags	Flags	**	**	**	**	0	No Loop
56	scaleTuning	cent/key	0	0	none	1200	oct/ky	100	semitone
57	exclusiveClass	arbitrary#	0	1	–	127	–	0	None
58	overridingRootKey	MIDI ky#	key# 0	0	lo key	127	hi key	-1	None

* Range depends on values of start, loop, and end points in sample header.

** Range has discrete values based on bit flags

WHAT IS CLAIMED IS:

1. A memory for storing audio sample data for access by a program being executed on a audio data processing system, comprising:

a data format structure stored in said memory, said data format structure including information used by said program and including

at least one preset, said preset referencing an instrument, said preset optionally including one or more articulation parameters for specifying aspects of said instrument;

at least one instrument referenced by each of said presets, each said instrument referencing an audio sample and optionally including one or more articulation parameters for specifying aspects of said instrument;

each of said articulation parameters being specified in units related to a physical phenomenon which is unrelated to any particular machine for creating or playing audio samples.

2. The memory of claim 1 wherein said units are perceptively additive.

3. The memory of claim 2 wherein said units are specified such that adding the same amount in such units to two different values in such units will proportionately affect the underlying physical values represented by said units, said units including percentages and decibels.

4. The memory of claim 2 wherein one of said units is absolute cents, wherein an absolute cent is 1/100 of a semitone, referenced to a 0 value corresponding to MIDI key number 0, which is assigned to 8.1758 Hz.

5. The memory of claim 4 wherein instrument articulation parameters expressed in absolute cents include:

modulation LFO frequency; and
initial filter cutoff.

6. The memory of claim 2 wherein one of said units is a relative time expressed in time cents, wherein timecents is defined for two periods of time T and U to be equal to 1200 $\log_2$ (T/U).

7. The memory of claim 6 wherein instrument articulation parameters expressed in relative time cents include:
modulation LFO delay;
vibrato LFO delay;
modulation envelope delay time;
modulation envelope attack time;
volume envelope attack time;
modulation envelope hold time;
volume envelope hold time;
modulation envelope decay time;
modulation envelope release time; and
volume envelope release time.

8. The memory of claim 1 wherein one of said units is an absolute time expressed in time cents, wherein timecents is defined for a time T in seconds to be equal to 1200 $\log_2$ (T).

9. The memory of claim 1 wherein instrument articulation parameters expressed in absolute time cents include:
modulation LFO delay;
vibrato LFO delay;
modulation envelope delay time;
modulation envelope attack time;
volume envelope attack time;
modulation envelope hold time;
volume envelope hold time;
modulation envelope decay time;

modulation envelope release time; and
volume envelope release time.

10. The memory of claim 1 wherein one or more of said audio samples comprise a block of data comprising:
one or more data segments of digitized audio;
a sample rate associated with each of said digitized audio segments;
an original key associated with each of said digitized audio segments; and
a pitch correction associated with said original key.

11. The memory of claim 1 wherein said articulation parameters comprise generators and modulators, at least one of said modulators comprising:
a first source enumerator specifying a first source of realtime information associated with said one modulator;
a generator enumerator specifying a one of said generators associated with said one modulator;
an amount specifying a degree said first source enumerator affects said one generator;
a second source enumerator specifying a second source of realtime information for varying said degree said first source enumerator affects said one generator; and
a transform enumerator specifying a transformation operation on said first source.

12. The memory of claim 1 wherein said audio samples include stereo audio samples, each of said stereo audio samples being a block of data including a pointer to a second block of data containing a mate stereo audio sample.

13. A memory for storing audio sample data for access by a program being executed on a audio data processing system, comprising:

a data format structure stored in said memory, said data format structure including information used by said program and including

a plurality of presets, each of said presets referencing an instrument, at least some of said presets including articulation parameters for specifying aspects of said instrument;

at least one instrument referenced by each of said presets, each of said instruments referencing an audio sample and including articulation parameters for specifying aspects of said instrument;

each of said articulation parameters being specified in units related to a physical phenomenon which is unrelated to any particular machine for creating or playing audio samples, said units being perceptively additive;

a plurality of said audio samples comprising a block of data including

one or more data segments of digitized audio,

a sample rate associated with each of said digitized audio segments,

an original key associated with each of said digitized audio segments, and

a pitch correction associated with said original key;

said articulation parameters comprising generators and modulators, at least one of said modulators including

a first source enumerator specifying a first source of realtime information associated with said one modulator,

a generator enumerator specifying a one of said generators associated with said one modulator,

an amount specifying a degree said first source enumerator affects said one generator,

a second source enumerator specifying a second source of realtime information for varying said degree said first source enumerator affects said one generator, and

a transform enumerator specifying a transformation operation on said first source.

14. The memory of claim 13 wherein said audio samples include stereo audio samples, each of said stereo audio samples being a block of data including a pointer to a second block of data containing a mate stereo audio sample.

15. An audio data processing system comprising:

a processor for processing audio sample data;

a memory for storing audio sample data for access by a program being executed on said processor, including:

a data format structure stored in said memory, said data format structure including information used by said program and including

at least one preset, each preset referencing at least one instrument, said presets optionally including one or more articulation parameters for specifying aspects of said instrument;

at least one instrument referenced by each of said presets, each of said instruments referencing an audio sample and optionally including one or more articulation parameters for specifying aspects of said instrument;

each of said articulation parameters being specified in units related to a physical phenomenon which is unrelated to any particular machine for creating or playing audio samples.

16. The system of claim 15 wherein said units are perceptively additive.

17. The system of claim 16 wherein said units are specified such that adding the same amount in such units to

two different values in such units will proportionately affect the underlying physical values represented by said units, said units including percentages and decibels.

18. The system of claim 16 wherein one of said units is absolute cents, wherein an absolute cent is 1/100 of a semitone, referenced to a 0 value corresponding to MIDI key number 0, which is assigned to 8.1758 Hz.

19. The system of claim 18 wherein instrument articulation parameters expressed in absolute cents include:
modulation LFO frequency; and
initial filter cutoff.

20. The system of claim 16 wherein one of said units is a relative time expressed in time cents, wherein timecents is defined for two periods of time T and U to be equal to 1200 $\log_2$ (T/U).

21. The system of claim 20 wherein preset articulation parameters expressed in time cents include:
modulation LFO delay;
vibrato LFO delay;
modulation envelope delay time;
modulation envelope attack time;
volume envelope attack time;
modulation envelope hold time;
volume envelope hold time;
modulation envelope decay time;
modulation envelope release time; and
volume envelope release time.

22. The system of claim 16 wherein one of said units is an absolute time expressed in time cents, wherein timecents is defined for a time T in seconds to be equal to 1200 $\log_2$ (T).

23. The system of claim 22 wherein instrument articulation parameters expressed in absolute time cents include:

modulation LFO delay;
vibrato LFO delay;
modulation envelope delay time;
modulation envelope attack time;
volume envelope attack time;
modulation envelope hold time;
volume envelope hold time;
modulation envelope decay time;
modulation envelope release time; and
volume envelope release time.

24. The system of claim 15 wherein a plurality of said audio samples comprise a block of data comprising:

one or more segments of digitized audio;

a sample rate associated with each of said digitized audio segments;

an original key associated with each of said digitized audio segments; and

a pitch correction associated with said original key.

25. The system of claim 15 wherein said articulation parameters comprise generators and modulators, at least one of said modulators comprising:

a first source enumerator specifying a first source of realtime information associated with said one modulator;

a generator enumerator specifying a one of said generators associated with said one modulator;

an amount specifying a degree said first source enumerator affects said one generator;

a second source enumerator specifying a second source of realtime information for varying said degree said first source enumerator affects said one generator; and

a transform enumerator specifying a transformation operation on said first source.

26. The system of claim 15 wherein said audio samples include stereo audio samples, each of said stereo audio samples being a block of data including a pointer to a second block of data containing a mate stereo audio sample.

27. An audio data processing system comprising:
a processor for processing audio sample data;
a memory for storing audio sample data for access by a program being executed on said processor, including:
a data format structure stored in said memory, said data format structure including information used by said program and including
a plurality of presets, each of said presets referencing an instrument, at least some of said presets including articulation parameters for specifying aspects of said instrument;
at least one instrument referenced by each of said presets, each of said instruments referencing an audio sample and including articulation parameters for specifying aspects of said instrument;
each of said articulation parameters being specified in units related to a physical phenomenon which is unrelated to any particular machine for creating or playing audio samples, said units being perceptively additive;
a plurality of said audio samples comprising a block of data including
one or more data segments of digitized audio,
a sample rate associated with each of said digitized audio segments,
an original key associated with each of said digitized audio segments, and

a pitch correction associated with said original key;

said articulation parameters comprising generators and modulators, at least one of said modulators including

a first source enumerator specifying a first source of realtime information associated with said one modulator,

a generator enumerator specifying a one of said generators associated with said one modulator,

an amount specifying a degree said first source enumerator affects said one generator,

a second source enumerator specifying a second source of realtime information for varying said degree said first source enumerator affects said one generator, and

a transform enumerator specifying a transformation operation on said first source.

28. A method for storing music sample data for access by a program being executed on a audio data processing system, comprising the steps of:

storing a data format structure in said memory, said data format structure including information used by said program and including

at least one preset, said preset referencing an instrument, said preset optionally including one or more articulation parameters for specifying aspects of said instrument;

at least one instrument referenced by each of said presets, each said instrument referencing an audio sample and optionally including one or more articulation parameters for specifying aspects of said instrument;

each of said articulation parameters being specified in units related to a physical phenomenon

which is unrelated to any particular machine for creating or playing audio samples.

29. The method of claim 28 further comprising the step of specifying said units to be perceptively additive.

30. The method of claim 28 further comprising the steps of storing a plurality of said audio samples as a block of data comprising:

one or more data segments of digitized audio;

a sample rate associated with each of said digitized audio segments;

an original key associated with each of said digitized audio segments; and

a pitch correction associated with said original key.

31. The method of claim 28 wherein said articulation parameters comprise generators and modulators, at least one of said modulators comprising:

a first source enumerator specifying a first source of realtime information associated with said one modulator;

a generator specifying a one of said generators associated with said one modulator;

an amount specifying a degree said first source enumerator affects said one generator;

a second source enumerator specifying a second source of realtime information for varying said degree said first source enumerator affects said one generator; and

a transform enumerator specifying a transformation operation on said first source.

32. The method of claim 28 wherein said audio samples include stereo audio samples, each of said stereo audio samples being a block of data including a pointer to a second block of data containing a mate stereo audio sample.

33. The method of claim 28 wherein at least one of said audio samples includes a loop start point and a loop end point, and further comprising the step of forcing proximal data points surrounding said loop start point and said loop end point to be substantially identical.

34. The method of claim 33 wherein the number of said substantially identical proximal data points is eight or less.

35. The memory of claim 1 wherein at least one of said audio samples includes a loop start point and a loop end point, and wherein proximal data points surrounding said loop start point and said loop end point are set to be substantially identical.

36. The memory of claim 35 wherein the number of said substantially identical proximal data points is eight or less.

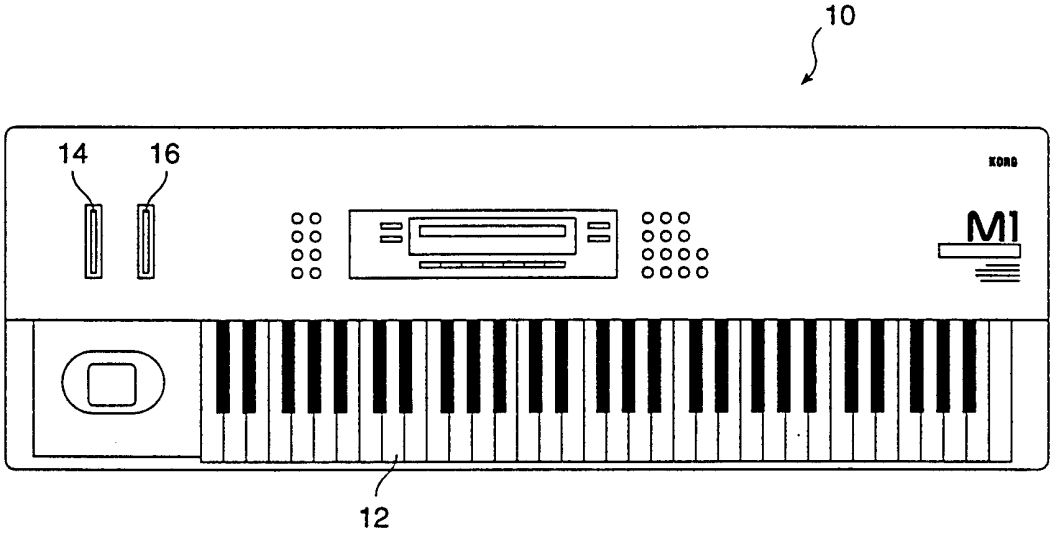

FIG. 1

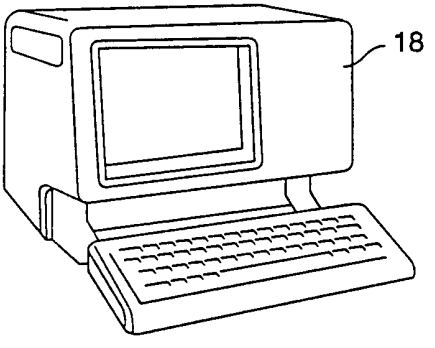

FIG. 2A

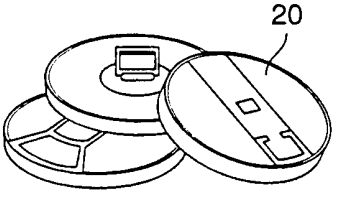

FIG. 2B

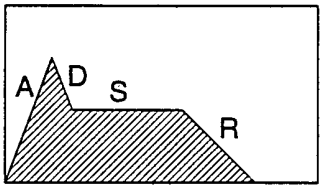

FIG. 4A

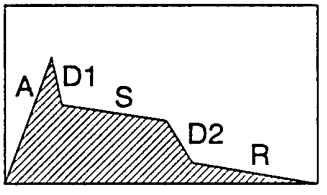

FIG. 4B

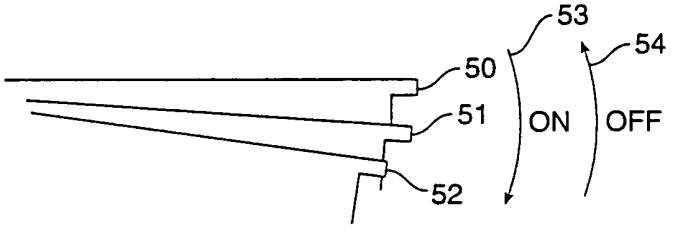

FIG. 5

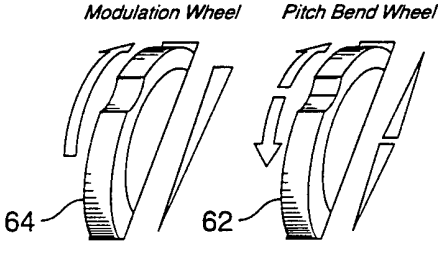

FIG. 6

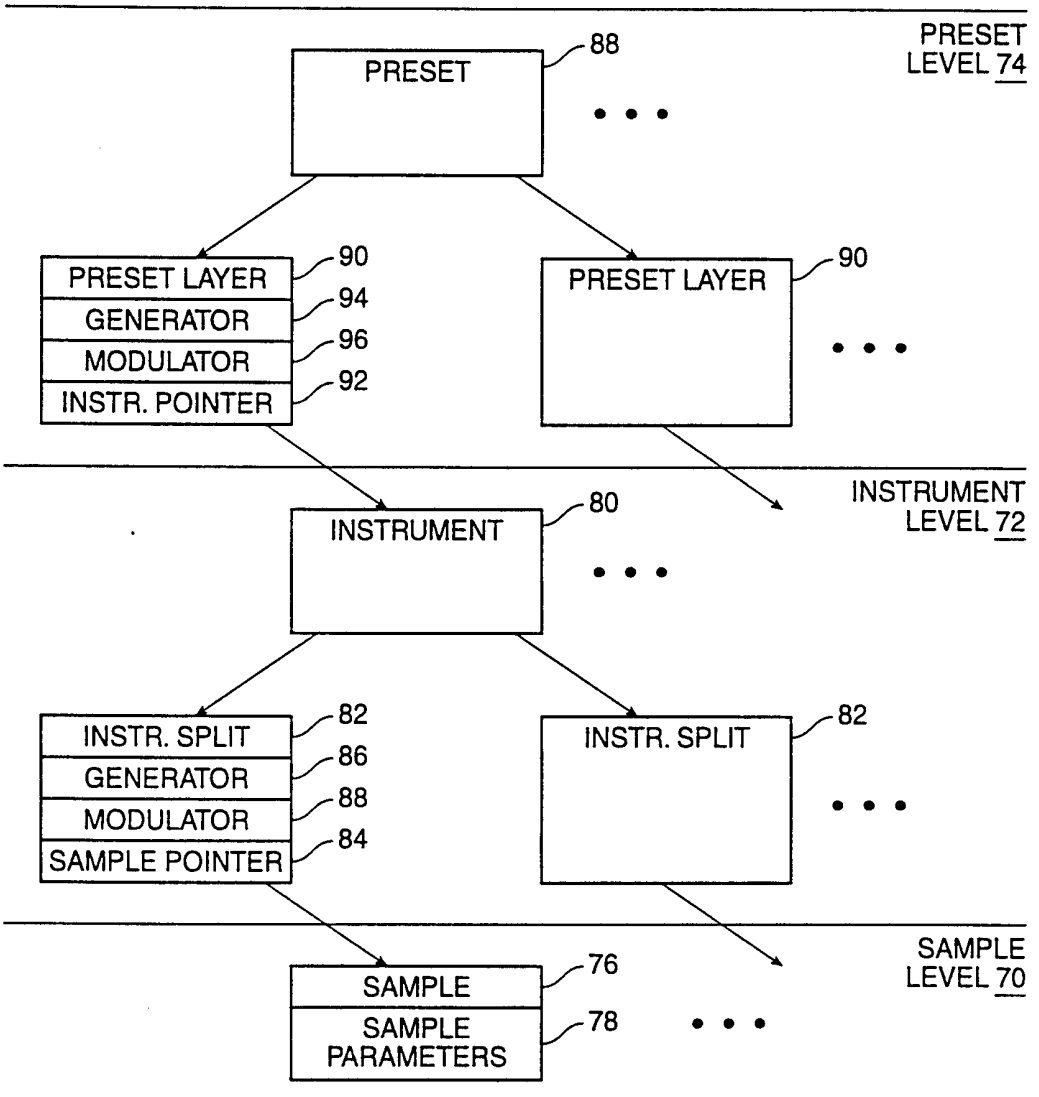
PRESET LEVEL 74
PRESET
88
PRESET LAYER
90
GENERATOR
94
MODULATOR
96
INSTR. POINTER
92
PRESET LAYER
90
INSTRUMENT LEVEL 72
INSTRUMENT
80
INSTR. SPLIT
82
GENERATOR
86
MODULATOR
88
SAMPLE POINTER
84
INSTR. SPLIT
82
SAMPLE LEVEL 70
SAMPLE
76
SAMPLE PARAMETERS
78

FIG. 7

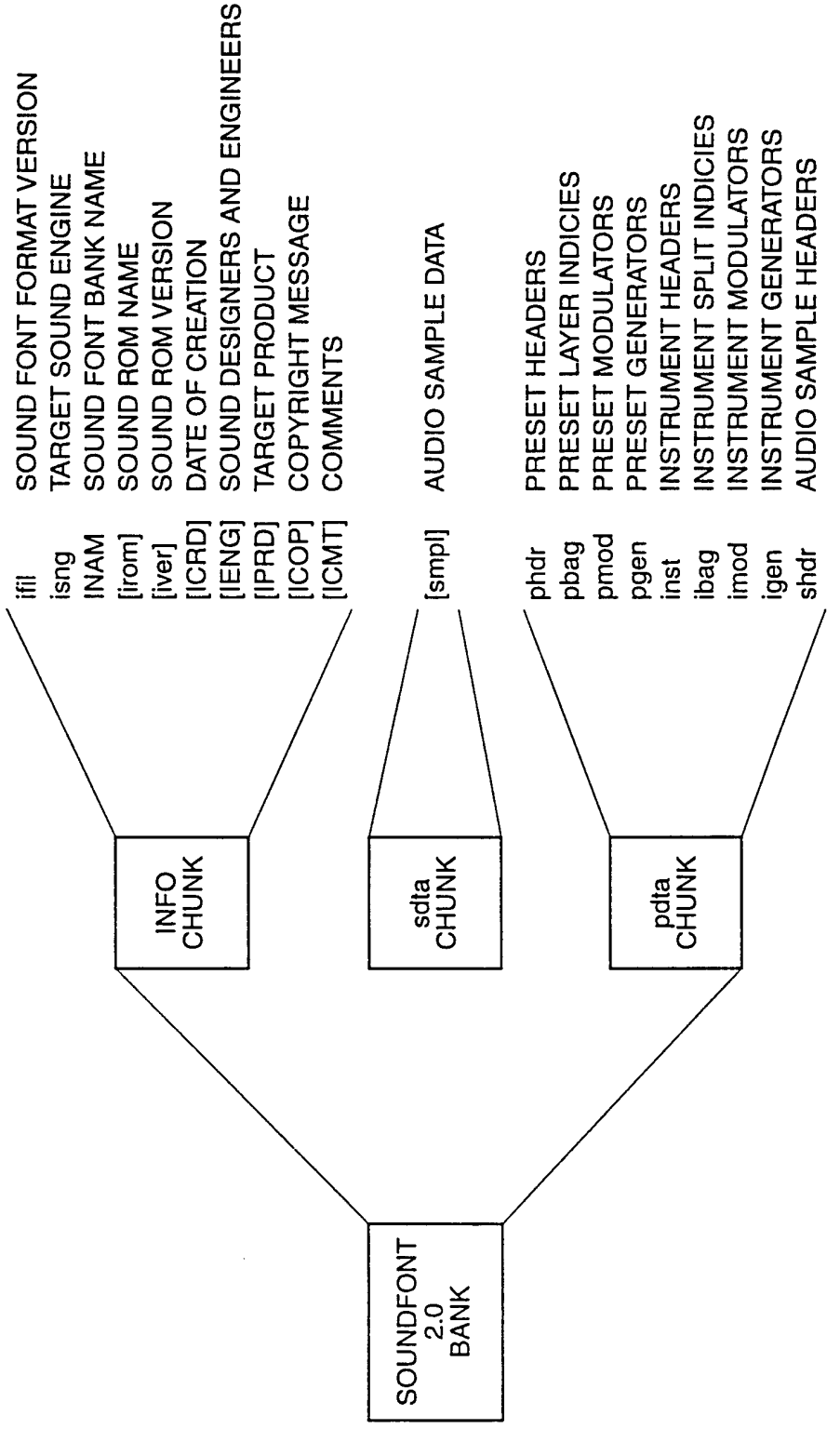
SOUNDFONT 2.0 BANK
INFO CHUNK
sdta CHUNK
pdta CHUNK
ifil SOUND FONT FORMAT VERSION
isng TARGET SOUND ENGINE
INAM SOUND FONT BANK NAME
[irom] SOUND ROM NAME
[iver] SOUND ROM VERSION
[ICRD] DATE OF CREATION
[IENG] SOUND DESIGNERS AND ENGINEERS
[IPRD] TARGET PRODUCT
[ICOP] COPYRIGHT MESSAGE
[ICMT] COMMENTS
[smpl] AUDIO SAMPLE DATA
phdr PRESET HEADERS
pbag PRESET LAYER INDICIES
pmod PRESET MODULATORS
pgen PRESET GENERATORS
inst INSTRUMENT HEADERS
ibag INSTRUMENT SPLIT INDICIES
imod INSTRUMENT MODULATORS
igen INSTRUMENT GENERATORS
shdr AUDIO SAMPLE HEADERS
LEVEL 0
LEVEL 1

FIG. 8

"RIFF" <size> "sfbk"
"LIST" <size> "INFO"
"ifil" <size> <format version>
"isng" <size> "<target sound engine"
"INAM" <size> "<bank name>"
["irom" <size> "<ROM name>"]
["iver" <size> "<ROM version>"]
["ICRD" <size> "<creation date>"]
["IENG" <size> "<names of engineers and sound designers>"]
["IPRD" <size> "<target product>"]
["ICOP" <size> "<copyright string>"]
["ICMT" <size> "<comment string>"]
[optional future INFO chunk sub-chunks]
"LIST" <size> "sdta"
["smpl" <size> <digital audio data>]
"LIST" <size> "pdta"
"phdr" <size> <preset headers>
"pbag" <size> <preset layer indicies>
"pmod" <size> <preset layer modulators>
"pgen" <size> <preset layer generators>
"inst" <size> <instrument headers>
"ibag" <size> <instrument split indicies>
"imod" <size> <instrument split modulators>
"igen" <size> <instrument split generators>
"shdr" <size> <sample headers>

FIG. 9

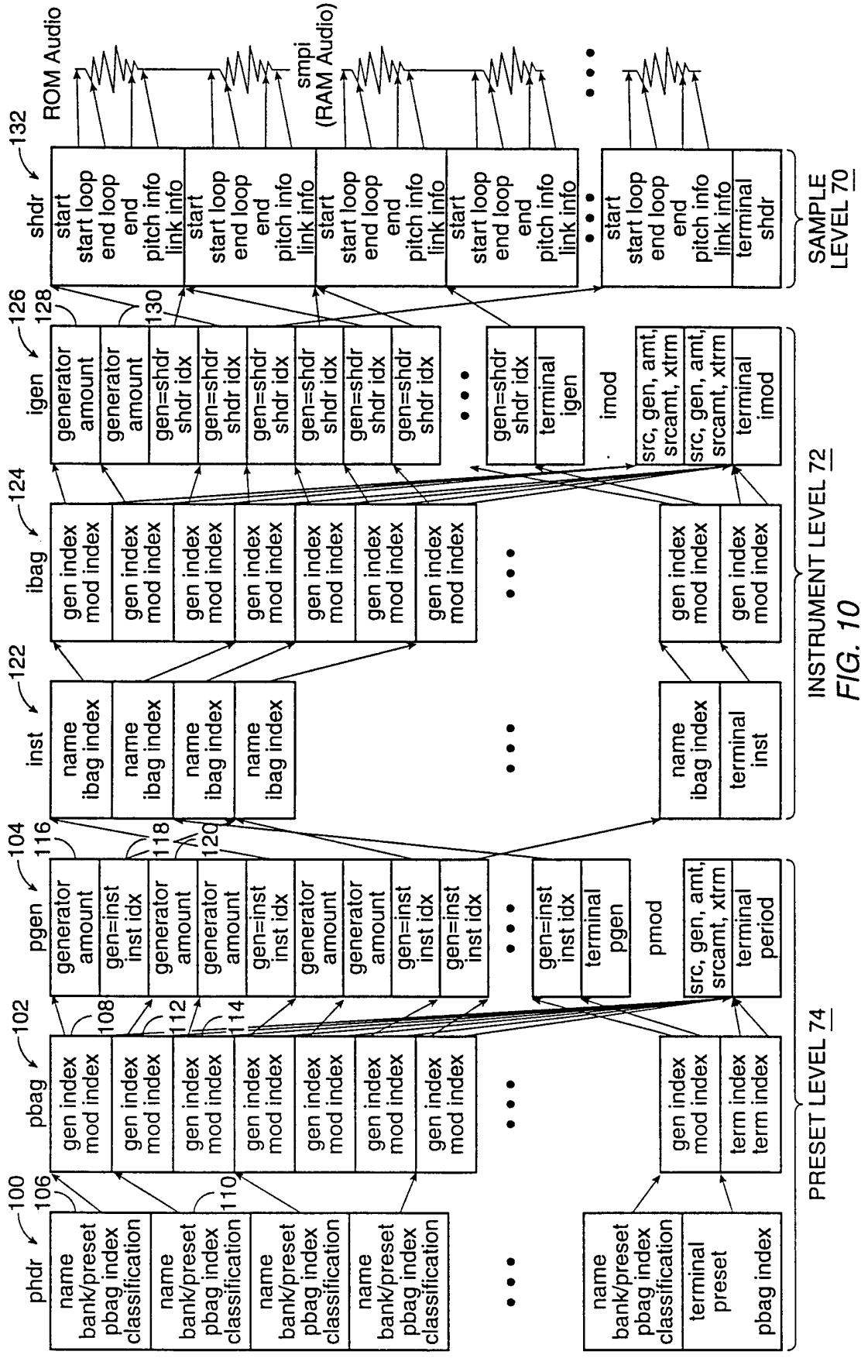
ROM Audio
smpi (RAM Audio)
132
shdr
start
start loop
end loop
end
pitch info
link info
terminal shdr
SAMPLE LEVEL 70
126
128
130
igen
generator amount
gen=shdr shdr idx
terminal igen
imod
src, gen, amt, srcamt, xtrm
terminal imod
124
ibag
gen index
mod index
122
inst
name
ibag index
terminal inst
INSTRUMENT LEVEL 72
104
116
118
120
pgen
gen=inst inst idx
terminal pgen
pmod
terminal period
102
108
112
114
pbag
term index
term index
100
106
110
phdr
name
bank/preset
pbag index
classification
terminal preset
pbag index
PRESET LEVEL 74

FIG. 10

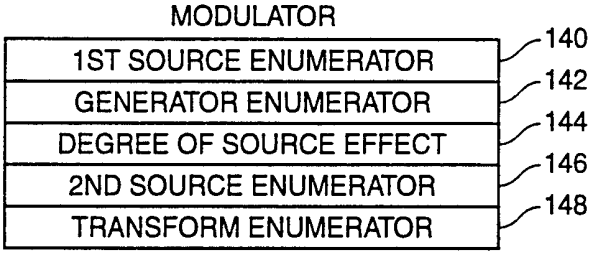

FIG. 11

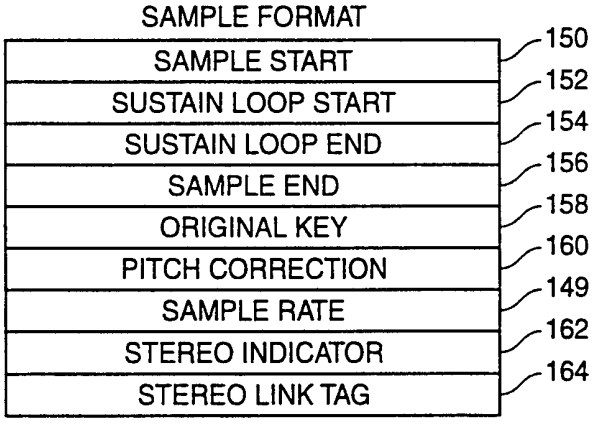

FIG. 12

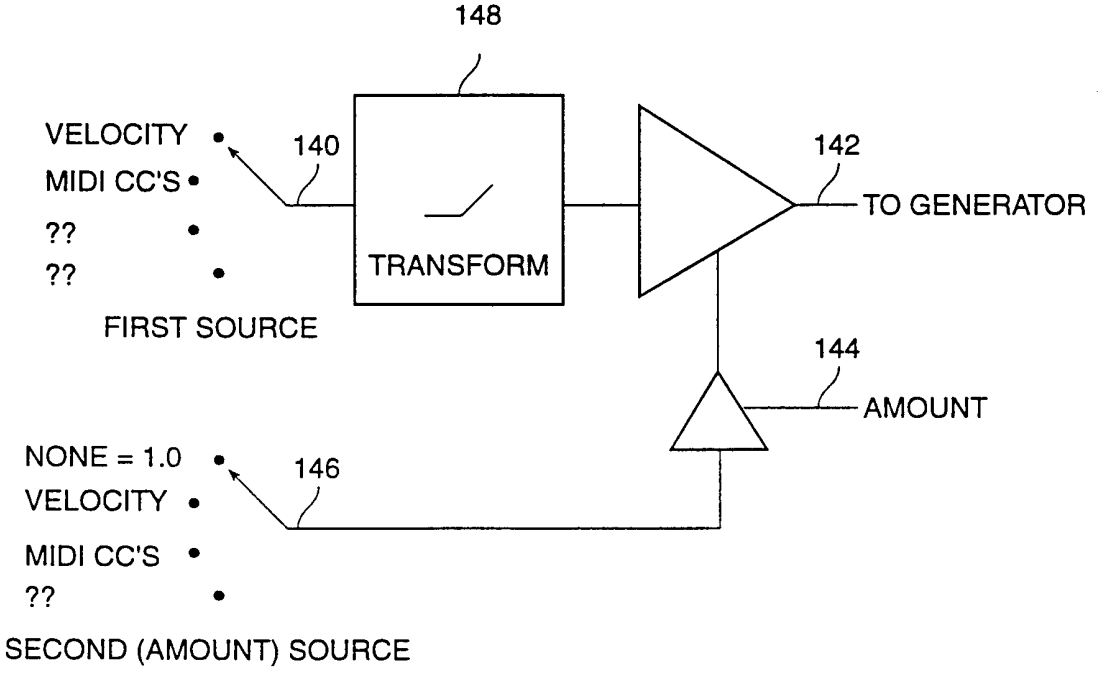

FIG. 13