RELEASE CANDIDATE DRAFT

Headspace
Rich Music Format (RMF):
The File Format

Document Version 0.08
Modified 11/9/98
Reflects HIRF Specification version 1

Contents

0. Essential Definition of 'RMF File'.....2

1. Supporting Definitions.....3

1.1. Headspace HIRF File Format.....3
1.1.0. Number Formats used in HIRF Files.....5
1.1.1. HIRF File Header.....6
1.1.2. HIRF Toplevel Resource Format.....7

1.2. HIRF Toplevel Resource Types.....11
1.2.1. Toplevel Resource `'SONG'`.....12
1.2.2. Toplevel Resource `'INST'`.....19
1.2.3. Toplevel Resource `'snd '`.....40
1.2.4. Toplevel Resource `'VERS'`.....54

2. An Example RMF File.....55

0. Essential Definition of ‘RMF File’

An RMF File is defined as any computer file (or machine-readable information pattern stored in any other information storage medium) that conforms to the Headspace Inter-Platform Resource File format (‘HIRF’) and includes exactly one HIRF Toplevel Resource of type `'SONG'`. HIRF *per se* is not the same as RMF.

Detailed supporting definitions of the HIRF format, and of HIRF Toplevel Resource Types including `'SONG'` follow.

1. Supporting Definitions

1.1. The Headspace Inter-Platform File Format (HIRF)

The Headspace Inter-Platform Resource File format (HIRF) was invented in order to furnish a structured data storage mechanism using the same data representation across all known computing platforms, as an important foundation for the platform-independent nature of the Headspace Audio Engine and its derived technologies.

The HIRF format consists of one HIRF File Header beginning at the start of the file, followed by a variable number of HIRF Toplevel Resources.

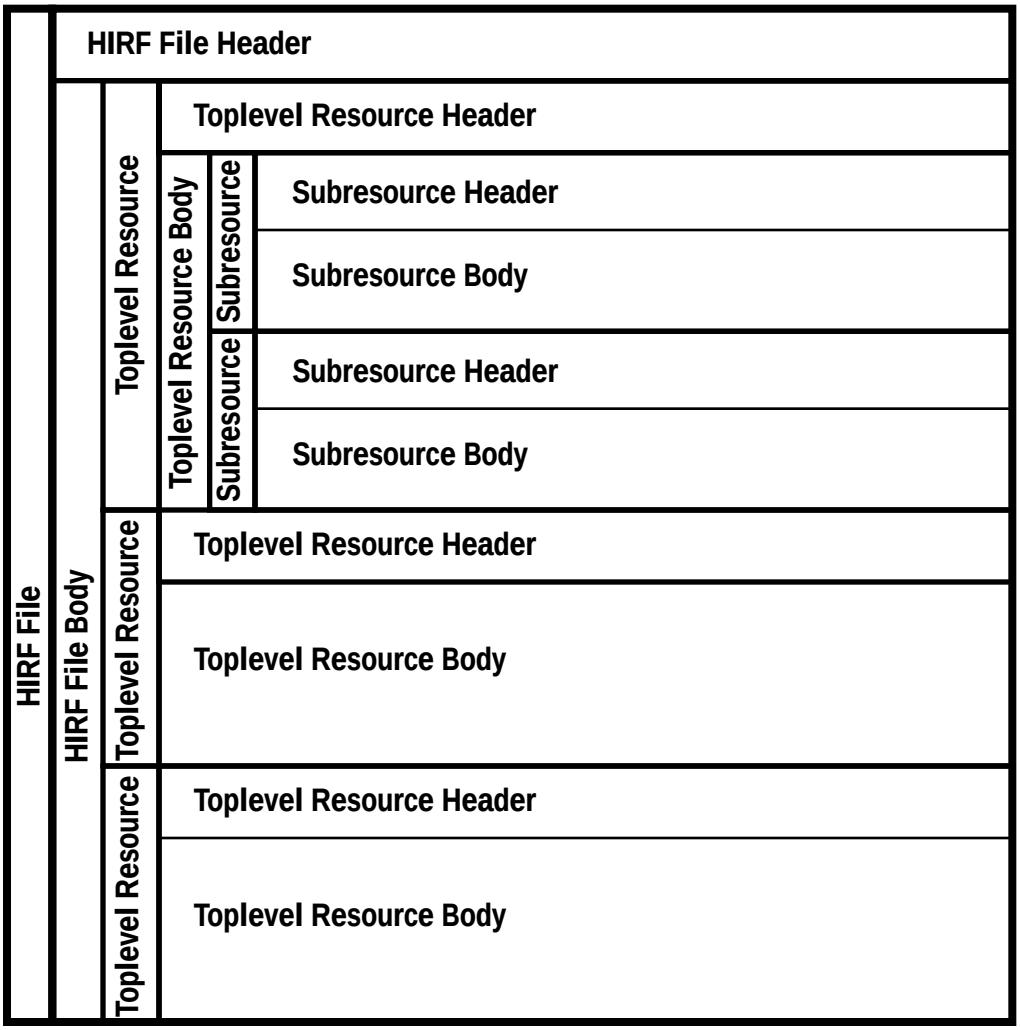
The HIRF File Header is always 12 bytes long, and HIRF Toplevel Resources are of variable length, but always at least 25 bytes long.

The number of HIRF Toplevel Resources that follow the HIRF File Header is indicated in field 3 of the HIRF File Header (see section 1.1.1). These Toplevel Resources follow the HIRF File Header immediately, starting at byte 12 of the file (0x000C). While there will usually be no gaps between HIRF Toplevel Resources, this is not guaranteed by the format specification; the only reliable way to move from one HIRF Toplevel Resource to the next is therefore to parse the HIRF Resource Headers (see section 1.1.2.1) and follow the chain of offsets stored in field 1 of each HIRF Toplevel Resource Header.

RELEASE CANDIDATE DRAFT

Advisory Note: The HIRF format uses a system of nested data chunks, each of which consist of a header followed by a variable number of content blocks. Because each of these content blocks may be in turn composed of a header and a variable number of subordinate content blocks, a HIRF-format file may exhibit a similar structure simultaneously at different levels of its nesting, and this can be confusing.

In this document we have attempted to reduce the likelihood of this confusion with diagrams and a careful choice of wording, but it may be helpful at the outset to take a moment to study this schematic visualization of the kind of nested structure that an HIRF File can exhibit:



For another visual representation of HIRF structure, see **2. An Example RMF File**.

1.1.0. Binary Number Formats Used in HIRF Files

In the HIRF File format, all binary integers longer than one byte are stored in Motorola byte order, i.e. with the high-order byte at the lowest address (or lowest offset) and the low-order byte at the highest address (or highest offset).

This is commonly known as ‘big endian’ representation, in contrast to ‘little-endian’ or Intel byte order.

1.1.1. HIRF File Header

Every HIRF-format file starts with a 3-field, 12-byte header. A HIRF-format file can be recognized by examining bytes 0 through 7 for the characters ‘IREZ’ and a 4-byte HIRF format version number.

(Byte numbers are relative to the start of this HIRF file)

Field	Bytes	Meaning and Data Format
1	0 - 3	The letters 'IREZ', expressed as four 1-byte ASCII characters. This is the HIRF file type identifier.
2	4 - 7	The version of the HIRF format specification to which the file conforms, expressed as an unsigned 32-bit binary integer. The HIRF specification is currently at version 1.
3	8 - 11	The number of HIRF Resources present in the file, expressed as an unsigned 32-bit binary integer, and counting from 1 (not from 0).

Example:

```
     HIRF File Header ---------------------------------------------
     |
0000:|    OSType('IREZ')  // File master chunk type
     |                    //   'I' 'R' 'E' 'Z' == 0x49 52 45 5A
0004:|    long(1)         // File conforms to HIRF format version 1
     |                    //   0x00000001
0008:|    long(3)         // 3 HIRF resources in this file
     |                    //   0x00000003
     |
     ----End of HIRF File Header -----------------------------------
```

1.1.2. HIRF Toplevel Resource Format

Every top-level Resource in a HIRF-format file consists of a HIRF Toplevel Resource Header, and usually a HIRF Toplevel Resource Body.

The HIRF Toplevel Resource Header is of variable length, but is always at least 25 bytes long; The length and format of the HIRF Toplevel Resource Body varies dramatically depending on the HIRF Toplevel Resource Type found in field 2 of the HIRF Toplevel Resource Header.

The Body for this HIRF Toplevel Resource follows immediately after field 6 of the HIRF Toplevel Resource Header, unless the length of the Body is zero; in that case, the HIRF Toplevel Resource Header of the next HIRF Toplevel Resource follows field 6 immediately; or, if this is the last HIRF Toplevel Resource in the file, the last byte of field 6 is also the last byte of the file.

The format for the HIRF Toplevel Resource Header and the several formats for the Bodies for the various HIRF Toplevel Resource Types are detailed hereunder.

1.1.2.1. HIRF Toplevel Resource Header

Every HIRF Toplevel Resource starts with a 6-field header of variable length.

(Byte numbers are relative to the start of this HIRF Toplevel Resource)

Field	Bytes	Meaning and Data Format
1	0 - 3	The position (i.e. offset from start of file) of the next HIRF Toplevel Resource in the file, expressed as a positive 32-bit unsigned binary integer. In the case of the last HIRF Toplevel Resource in the file, this field should contain one more than the file offset of the last byte in the file.
2	4 - 7	The HIRF Toplevel Resource Type identifying the format of the data in the Body of this HIRF Toplevel Resource, expressed as four 1-byte ASCII characters. For example, one HIRF Toplevel Resource Type used in all RMF files is `'SONG'`; another one used in some RMF files is `'ecmi'`.
3	8 - 11	The ID number of this HIRF Toplevel Resource, expressed as an unsigned 32-bit binary integer. Within the HIRF file, this ID Number should be unique for this HIRF Toplevel Resource Type; however, the same ID number can be used for HIRF Toplevel Resources with different HIRF Toplevel Resource Types.
4[1]	12	The Length of the Name in field 5, expressed as an unsigned 8-bit binary integer.
5[1]	n bytes starting at byte 13	The Name of this HIRF Toplevel Resource, expressed as a sequence of 1-byte ASCII characters (no terminating zero). Within this HIRF file, this Name should be unique for this HIRF Toplevel Resource Type, but the same Name can be used in HIRF Toplevel Resources with different HIRF Toplevel Resource Types. Note that HIRF Toplevel Resource Names inhabit a separate namespace from HIRF Toplevel Resource Types.
6	4 bytes following field 5	The Length in bytes of the Body of this HIRF Toplevel Resource, expressed as an unsigned 32-bit binary integer.

[1] Fields 4 and 5 together can alternatively be considered a single ‘Pascal string’ field; also, if there is no name for this HIRF Resource, field 4 contains zero and field 6 starts at Byte 13.

Example:

```
      |    HIRF Toplevel Resource Header ----------------------------
      |    |    long(0x2E30)    // Next Toplevel Resource starts at
      |    |                    //    file offset 0x2E30
      |    |                    //   0x00002E30
0010: |    |    OSType('ecmi')  // Resource Body contains 'ecmi' data
      |    |                    //   'e' 'c' 'm' 'i' == 0x65 63 6D 69
0014: |    |    long(0x3530)    // This is 'ecmi' Toplevel Resource
      |    |                    //   no. 0x3530:
      |    |                    //   0x00003530
0018: |    |    byte(11)        // The name of this 'ecmi' Resource
      |    |    "modern rock"   //   is "modern rock", expressed as
      |    |                    //   a "Pascal-string":
      |    |                    //     0x0B - Length is 11 characters
      |    |                    //   And then the 11 characters:
      |    |                    //     0x6D 6F 64 65 72 6E 20 72
      |    |                    //     0x6F 63 6B
0024: |    |    long(0x2E08)    // The Body of this 'ecmi' Resource
      |    |                    //   is 0x2E08 bytes long
      |    |                    //   0x00002E08
      |    ----End of HIRF Toplevel Resource Header -----------------
```

1.1.2.2. HIRF Toplevel Resource Body

The format and length of a HIRF Toplevel Resource Body depends upon the HIRF Toplevel Resource Type that appears in its HIRF Toplevel Resource Header.

Headspace has defined a number of different HIRF Toplevel Resource Types, each with their own HIRF Toplevel Resource Body format. For enumeration and descriptions of these formats, see section **1.2. HIRF Toplevel Resource Types**.

1.2. HIRF Toplevel Resource Types

Note: For purposes of defining the term ‘RMF File’, the most relevant HIRF Toplevel Resource Types are `'SONG'` and `'INST'`, which are described in sections 1.1.3.1. and 1.1.3.2.; we expect future versions of this document to detail the remaining Types.

Currently defined HIRF Toplevel Resource Types include:

Type	See Section	Description
`'SONG'`	1.2.1	Song
`'INST'`	1.2.2	Instrument format
`'Midi'` `'MIDI'`	—	Standard MIDI File data (The Standard MIDI File format standard is administered by the International MIDI Association; see www.ima.com)
`'cmid'`	—	Standard MIDI File data, compressed with Headspace’s Beatnik Compression technique
`'emid'`	—	Standard MIDI File data, encrypted
`'ecmi'`	—	Standard MIDI File data, compressed with Headspace’s Beatnik Compression technique, and then encrypted
`'snd '`	1.2.3	Digital Audio Sample
`'csnd'`	—	Digital Audio Sample, compressed with Headspace’s Beatnik Compression technique
`'esnd'`	—	Digital Audio Sample, encrypted
`'vers'`	1.2.4	File version designator

1.2.1. Toplevel Resource Type `'SONG'`

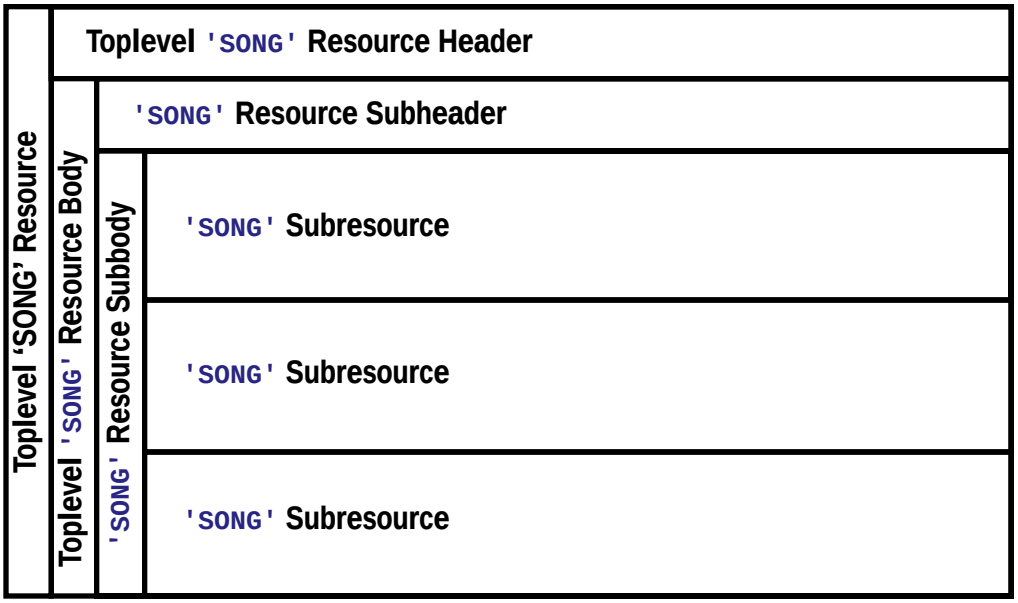
A `'SONG'` Toplevel Resource represents a musical composition or performance, stored either as a sequence of note events (as in a Standard MIDI File), as a digital audio recording, or as a combination of both, and ties the note data together with additional information related to its use and playback. Note that no MIDI note event data nor sample data is stored directly within the Toplevel `'SONG'` Resource; rather, the `'SONG'` Resource contains a reference to other Toplevel Resource in the same HIRF File that does holds the musical data.

Every Toplevel `'SONG'` Resource starts with a HIRF Toplevel Resource Header (see section 1.1.2.1), followed by a HIRF Toplevel Resource Body (described hereunder).

The HIRF Toplevel Resource Body in turn consists of a `'SONG'` Resource Subheader, and usually a `'SONG'` Resource Subbody. If a `'SONG'` Resource Subbody is present, it follows its `'SONG'` Resource Subheader immediately, and in turn consists of a variable number of `'SONG'` Subresources.

The `'SONG'` Resource Subheader is always 50 bytes long, and the `'SONG'` Resource Subbody is of variable length; some of the `'SONG'` Subresources have fixed length, but most are of variable length.

Schematically, a `'SONG'` Resource looks like this (note, though, that the number of Subresources can vary):



1.2.1.1. `'SONG'` Resource Subheader

The Body of every HIRF Toplevel `'SONG'` Resource starts with a 15-field Subheader:

(Byte numbers are relative to the start of this `'SONG'` Resource Subheader)

Field	Bytes	Meaning and Data Format
1	0 - 1	ID Number of the HIRF Toplevel Resource containing the MIDI music composition/performance data to use for this `'SONG'` at playback time, expressed as a 16-bit unsigned binary integer. This number must match the ID Number field of a HIRF Toplevel Resource with MIDI data in this same HIRF-format file and containing playable MIDI music composition/performance data. (When converting this 16-bit sample number to a 32-bit HIRF Toplevel Resource ID for purposes of searching for the sample resource, the high 2 bytes will be set to 0x0000, and the resulting Resource ID will be sought in HIRF Toplevel Resource Types `'Midi'`, `'MIDI'`, `'cmid'`, `'emid'`, and `'ecmi'`.
2	2	(This byte is reserved for future uses and should be ignored)
3	3	Number of the Reverb Type to be used when playing this `'SONG'`, expressed as an 8-bit unsigned binary integer. Headspace currently defines the following Reverb Type numbers: 0: Use current reverb (don't change) 1: Dry (no reverb) 2: Closet 3: Garage 4: Lab 5: Cavern 6: Dungeon 7: Small reflections 8: Early reflections 9: Basement 10: Banquet hall 11: Catacombs

4	4 - 5	Factor by which to scale the Tempo stored with the music composition / performance data for this song (as indicated by field 1) at playback time. For example, a factor of 2.0 and a stored tempo of 85 beats per minute would yield an effective playback tempo of 170 beats per minute. This Tempo Factor is expressed in an unusual format: a 16-bit fixed-point fractional number, with the bit pattern 0x411B (decimal 16,667) meaning a factor of 1.0. (As another example of the number format, 0.5 would be expressed as 0x208D or decimal 8333.)
5	6	Format of the MIDI music composition/performance data for this `'SONG'` (i.e. to be found in the HIRF Toplevel Resource indicated by field 1 of this Subheader), expressed as an 8-bit unsigned binary integer: 0: (reserved – a Headspace private format) 1: RMF Structured data (local file) 2: RMF Linear data (streaming data)
6	7	Flag indicating whether or not the Subresources and music composition/performance data resource for this `'SONG'` are encrypted, expressed as a 1-bit binary flag stored in the low bit of a single byte. 0: Unencrypted 1: Encrypted
7	8 - 9	Number of musical semitones (or 'half-steps') by which to Transpose the Notes in the MIDI music composition / performance data for this song (as indicated by field 1) at playback time, expressed as a 16-bit signed binary integer. For example, a 7 in this field would cause the song to be transposed up by a perfect fifth.
8	10 - 11	Maximum number of simultaneous digital audio files and digital audio streams that the MIDI music composition / performance data for this song (as indicated by field 1) may cause to be played, expressed as a 16-bit unsigned binary integer. This information is used at playback time to configure the Headspace Audio Engine.

9	12 - 13	Maximum number of simultaneous voices that the MIDI music composition / performance data for this song (as indicated by field 1) may cause to sound on the playback music synthesizer, expressed as a 16-bit unsigned binary integer. This information is used at playback time to configure the Headspace Audio Engine.
10	14 - 15	Maximum number of simultaneous full-scale audio signals that the MIDI music composition / performance data for this song (as indicated by field 1) may cause to occur, expressed as a 16-bit unsigned binary integer. This information is used at playback time to configure the Headspace Audio Engine.
11	16 - 17	Volume to use for this song at playback time, expressed in an unusual format: a 16-bit fixed-point fractional number, with the bit pattern 0x007F (decimal 127) meaning a 'normal' volume of 100%. (As another example of the number format, 200% of normal would be expressed as 0x00FE or decimal 254.) A value of 0 in this field is interpreted as a token to use the normal playback volume (100%).
12	18	Flag indicating whether or not this Toplevel `'SONG'` Resource appears within an instrument bank, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0 The `'SONG'` **is not** in an instrument bank 1 The `'SONG'` **is** in an instrument bank
13	19	(This byte is reserved for future uses and should be ignored.)
14	20 - 47	(These 28 bytes, equivalent to seven 32-bit long integers, are reserved for future uses and should be ignored.)
15	48 - 49	The number of Subresources that follow in the `'SONG'` Resource Body (see section 1.2.1.2.), expressed as a 16-bit unsigned binary integer and counting from 1 (rather than 0).

Example:

```
2E50: |     |     'SONG' Resource Subheader -----------------------
      |     |     |
      |     |     |     short(0x3530)    // Use music data in Toplevel
      |     |     |                      // Resource ID # 0x3530 to play
      |     |     |                      //    this 'SONG'
      |     |     |                      //    0x3530
2E52: |     |     |     byte(0)          // Reserved byte
      |     |     |                      //    0x00
2E53: |     |     |     byte(4)          // Use reverb type 4 ('Lab') when
      |     |     |                      //    playing this song
```

```
      |     |     |                    //   0x04
2E54: |     |     |    short(0)        // Use song tempo stored in the
      |     |     |                    //   music data
      |     |     |                    //   0x0000
2E56: |     |     |    byte(1)         // Music data in Toplevel
      |     |     |                    //   Resource # 0x3530 is in
      |     |     |                    //   RMF Structured format
      |     |     |                    //   0x01
2E57: |     |     |    byte(1)         // Resources for this 'SONG' are
      |     |     |                    //   encrypted
      |     |     |                    //   0x01
2E58: |     |     |    short(0)        // Don't transpose music data
      |     |     |                    //   0x0000
2E5A: |     |     |    short(4)        // Maximum of 4 digital audio
      |     |     |                    //   files and streams at once
      |     |     |                    //   0x0004
2E5C: |     |     |    short(28)       // Maximum of 28 music synth
      |     |     |                    //   notes at once:
      |     |     |                    //   0x001C
2E5E: |     |     |    short(8)        // Allow 8 full-scale signals
      |     |     |                    //   before clipping
      |     |     |                    //   0x0008
2E60: |     |     |    short(0)        // Play music data at normal
      |     |     |                    //   (full) volume
      |     |     |                    //   0x0000
2E62: |     |     |    byte(0)         // This 'SONG' not embedded in an
      |     |     |                    //   instrument bank
      |     |     |                    //   0x00
2E63: |     |     |    byte(0)         // Reserved byte
      |     |     |                    //   0x00
2E64: |     |     |    long[7]         // 7 longs of reserved space here:
      |     |     |                    //   0x00000000 00000000
      |     |     |                    //   0x00000000 00000000
      |     |     |                    //   0x00000000 00000000
      |     |     |                    //   0x00000000
2E80: |     |     |    short(11)       // 11 Subresources follow
      |     |     |                    //   immediately in the Body
      |     |     |                    //   0x000B
      |     |     |
      |     |     ----End of 'SONG' Resource Subheader -------------
```

1.2.1.2. 'SONG' Resource Subbody

The Subbody of a `'SONG'` Resource consists of an uncounted list of one or more `'SONG'` Subresources. The `'SONG'` Subresources within this Subbody follow one another directly, with no gaps in between, and the number of Subresources that appear can be found in field 15 of the `'SONG'` Resource Subheader (see section 1.2.1.1.).

Each `'SONG'` Subresource in turn consists of two fields: a 4-character `'SONG'` Subresource Type and a variable-length data field. Most of these `'SONG'` Subresource Types are for the purpose of tagging the `'SONG'` with textual information related to the licensing and control of copyright-protected music compositions and sound recordings. These text Subresources may be either in clear text or encrypted, per field 6 of the Subheader (see section 1.2.1.1.). The two non-text Subresource types are `'RMAP'` and `'VELC'`, which are described within the following table.

(Byte numbers are relative to the start of this `'SONG'` Subresource)

Field	Bytes	Meaning and Data Format
1	0 - 3	A `'SONG'` Subresource Type identifying the function of the following parameter field, expressed as four 1-byte ASCII characters. Headspace currently defines the following `'SONG'` Subresource Types: `'TITL'` Title of the song `'PERF'` Name of the artist who performed the song `'COMP'` Name of the composer of the song `'COPD'` Date of copyright `'COPL'` Text of copyright notice `'LICC'` Contact information for licensing person `'LUSE'` Uses for which the song is licensed `'LDOM'` Internet domain name for which the song is licensed `'LTRM'` Term for which the song is licensed `'EXPD'` Expiration date of license `'NOTE'` General-purpose text for annotations `'INDX'` Index Number `'GENR'` Musical Genre of the song `'SUBG'` Musical Sub-Genre of the song `'RMAP'` MIDI Instrument number Patch Mapping (see description in field 2) `'VELC'` MIDI Note Velocity Map (see description in field 2)

2	n bytes starting at byte 4	This is the parameter field, the interpretation of which depends upon the `'SONG'` Subresource Type in field 1: • For `'RMAP'` Subresources: At playback time, any appearances in the associated music composition / performance data (see field 1 in the `'SONG'` Resource Subheader, section 1.2.1.1.) of field 2's 'from' value as a MIDI Program number is treated as though it were field 2's 'to' value. Field 2 contains both a 2-byte 'from' value and a 2-byte 'to' value, each expressed as a 16-bit unsigned binary integer. This yields a total of 4 bytes in field 2, and a total of 8 bytes in the whole Subresource. • For `'VELC'` Subresources: At playback time, remaps the velocity parameter of all MIDI Note On events appearing in the associated music composition / performance data to any desired value, by means of a direct lookup in the 128-word table provided in field 2. (MIDI note numbers range from 0 - 127). Field 2 contains a 128-entry array of 16-bit unsigned binary integers, for a total of 256 bytes in field 2 and a total of 260 bytes in the whole Subresource. Note that whereas MIDI velocities are in the range 0 - 127 (and the Headspace Audio Engine treats that as a normal range), the entries in this table are permitted to range as high as 0 - 508, which is 4 times the size of the MIDI velocity range. • For all other Subresource Types: Field 2 contains a character string with textual information whose meaning is indicated the `'SONG'` Subresource Type. If this is an unencrypted `'SONG'` Toplevel Resource (see field 6 of the `'SONG'` Resource Subheader) then free-text characters start at Byte 4 and run until a terminating zero byte is found. If, on the other hand, this is an encrypted `'SONG'` Resource, then the encrypted text starts at Byte 4 and the characters and terminating zero can only be recovered by applying the string decryption algorithm.

Example:

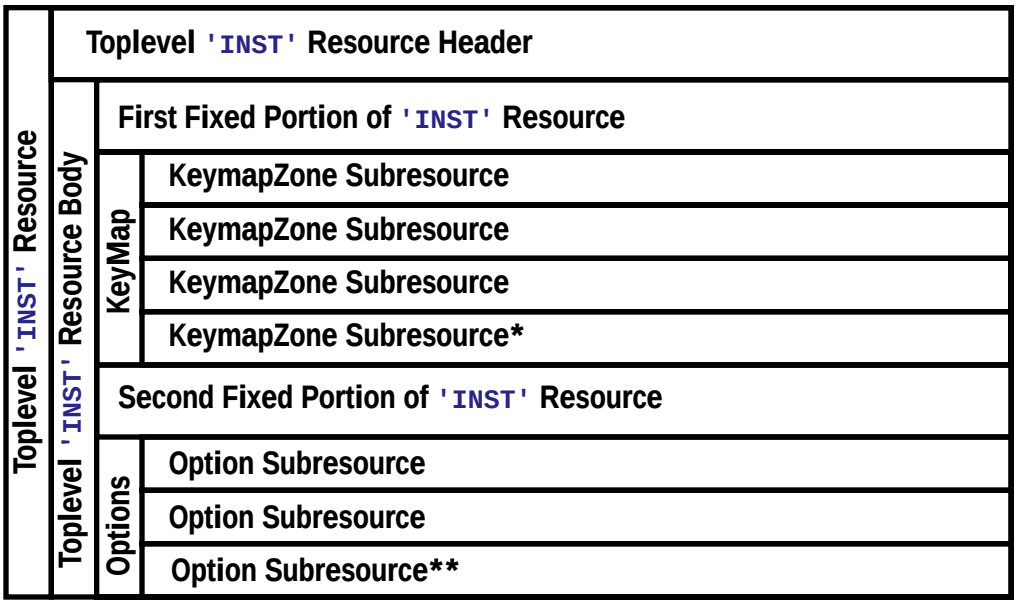
```
     |    |    |    'SONG' Subresource ----------------------------
     |    |    |    |    // 'TITL': Song Title text subresource
2E82:|    |    |    |    'T' 'I' 'T' 'L' == 0x54 49 54 4C
     |    |    |    |
     |    |    |    |    // encrypted string ("modern rock")
2E86:|    |    |    |    0x91 F8 FB 32 15 9D FA 4A 28 A4 16 64
     |    |    |    ----End of Song Subresource ---------------
```

1.2.2. Toplevel Resource Type `'INST'`

An `'INST'` Toplevel Resource represents a single machine-independent software musical instrument definition, as a MIDI Program Change event would select, for use at playback time by the music synthesizer in the Headspace Audio Engine (HAE) and derived technologies.

The Body of every HIRF Toplevel Resource with a Resource Type of `'INST'` consists of 13 fields, some of them of variable length and containing a variable number of Subresources; the first 7 fields in the Body are always in bytes 0 through 11, and the remainder needs to be parsed because of a number of optional and variable-length fields whose formats are described hereunder. Note that several fields in the `'INST'` Resource format are reserved, some of which are dictated by legacy issues.

Schematically, an `'INST'` Resource looks like this:



*From 0 to 8 KeymapZone Subresources here
**Any number of Option Subresources here

Note: Some of the following field descriptions are lengthy; this is necessary to explain the HAE synthesizer model in enough detail to understand the purpose of the data.

(Byte numbers are relative to the start of this Toplevel `'INST'` Resource Body)

Field	Bytes	Name	Meaning and Data Format
1	0 - 1	singleSampleNum	An instrument can either use a single sample for all MIDI note numbers ('single-sample instrument'), or it can assign one or more samples to different ranges of contiguous MIDI note numbers by means of a keyMap ('multi-sample instrument'). For a single-sample instrument, put the sample resource's HIRF Toplevel Resource ID here, expressed as a 16-bit unsigned binary integer, and put the word 0x0000 in the keyMap (field 8). (When converting this 16-bit sample number to a 32-bit HIRF Toplevel Resource ID for purposes of searching for the resource, the high 2 bytes will be set to 0x0000, and the resulting Resource ID will be sought for in HIRF Resource Types `'csnd'`, `'esnd'` and `'snd '`, in that order.) Or, for a multi-sample instrument, provide a keyMap in field 8 (see below) and field 1 will be ignored; however, we suggest you duplicate the sample resource ID from the first keymapZone in your keyMap (see field 8) here.
2	2 - 3	rootKeyOffset	Number of MIDI note numbers by which to offset the MIDI rootKey field stored with each sample resource used by this instrument. This offset is expressed as a 16-bit signed binary integer and should be in the MIDI note number range of 0 - 127 (with 60 equivalent to 'middle C'). This field should ordinarily be set to 0. This rootKeyOffset is provided to facilitate creative reuse of sample resources by tuning them to pitch ranges other than their original range. **Note:** This field is interpreted differently if the miscParmsMode bit in the INSTflags (field 4) is set to 0 (see **INSTflag Bitfields**, section 1.2.2.1). **Note:** This field is ignored for multi-sample instruments if the miscParmsMode bit in the INSTflags (field 4) is set to 1.
3	4	panPot	Initial stereo pan placement for each note played with this instrument, with -63 meaning hard left, 0 meaning center; and 63 meaning hard right.

4	5 - 6	INSTflags	Collection of several 1-bit flags governing various instrument modes and settings, including variations on how the `'INST'` resource itself should be interpreted, as detailed below under **INSTflag Bitfields** (section 1.2.2.1.).
5	7	(reserved)	Set this byte to 0, always.
6	8 - 9	rootKey[1]	For a single-sample instrument, the value in this field overrides the rootKey field stored in the sample resource indicated in field 1. This override is provided to facilitate creative simultaneous reuse of sample resources in multiple instruments by tuning them to pitch ranges other than their natural one. The rootKey is expressed as a 16-bit unsigned binary integer, and should be in the MIDI note number range of 0 - 127 (with 60 equivalent to 'middle C'). For a multiple-sample instrument, this field is ignored and should be set to 0.
7	10 - 11	volume[1]	For a single-sample instrument, the value in this field overrides the volume field stored in the sample resource indicated in field 1. This override is provided to help balance instruments relative to one another. The volume is expressed as a 16-bit unsigned binary integer, with 100 equivalent to a 'normal' volume of 100%. For a multiple-sample instrument, this field is ignored and should be set to 0.
		[1] Note: When the miscParmsMode bit in the INSTflags field is set to 0, these fields are interpreted differently; see **Rescaling Sample Data**, section 1.2.2.2.	

8	n bytes starting at byte 12	keyMap	For a multiple-sample instrument, a keyMap in this location specifies the sample resource to use for any given MIDI note number. A keyMap is a counted list of keymapZone Subresources, each of which connects a contiguous range of MIDI note numbers to a specific sample resource number. MIDI notes not falling into any of the provided keymapZones will not sound. The keymapZone Subresource format is described below in section 1.2.2.3. The first two bytes in this field must contain the number of keymapZone Subresources in this keyMap, expressed as a 16-bit unsigned binary integer; then the keymapZones follow immediately. For a single-sample instrument, this field must be set to 0x0000.
9	6 bytes following field 8	(reserved)	Set to 0x0000 0x8000 0x0000 always (don't miss the 0x8000 in the middle!)
10	n bytes following field 9	copyrightNotice	Text of an optional Copyright Notice for the instrument, expressed as a Pascal string (first byte is the number of characters, followed by the sequence of 1-byte ASCII characters that comprise the text, without any terminating NULL character).
11	n bytes following field 10	authorCredit	Text of an optional Author Credit for the INSTrument, expressed as a Pascal string (first byte is the number of characters, followed by the sequence of 1-byte ASCII characters that comprise the text, without any terminating NULL character).
12	12 bytes following field 11	(reserved)	Set all bytes to 0.
13	n bytes following field 12	optionsList	Option Subresources are used to describe a wide range of instrument features; each Option Subresource enables one such feature for this instrument. For an instrument that doesn't require any Options, put the word 0x0000 here. Or, to use Options, put a counted list of Option Subresources here, as required by your instrument design, and as described below in section 1.2.2.4.

1.2.2.1. INSTflags Bitfields

The meanings of the individual bitflags within the 1-bit INSTflags field (i.e. field 4 of a Toplevel `'INST'` Resource; see section 1.2.2.) are as follows:

(Bit 0 is the low-order bit, bit 16 is the high-order bit.)

Bit	Name	Meaning and Data Format
0	(reserved)	Set to 0.
1	(reserved)	Set to 0.
2	notPolyphonic	(under development)
3	miscParmsMode	How to interpret rootKey and volume fields: 0: As the sampleMultiplier and sampleDivisor for a Rescale Samples operation (applies for 8-bit mono samples only; see section 1.2.2.2.) 1: Normally, i.e. as rootKey and volume (applies to extendedFormat INST's only) **Note:** It is invalid to set both the miscParmsMode bit and the rescaleSamples bit to 1 in the same instrument.
4	rescaleSamples	Adjust sample data volume upon loading?: 0: No – Don't rescale (normal) 1: Yes – Rescale sample data (applies for 8-bit mono samples only; see section 1.2.2.2). **Note:** It is invalid to set both the miscParmsMode bit and the rescaleSamples bit to 1 in the same instrument.
5	(reserved)	Set to 0.
6	neverTranspose	Disable note transposition for voice using this instrument?: 0: No (normal for most pitched instruments) 1: Yes (normal for drums & 1-note keyMaps)
7	(reserved)	Set to 0.
8	noReverb	Mute reverb send for voices using this instrument?: 0: No – Allow reverb 1: Yes – Never reverb

9	extendedFormat	Is this `'INST'` Resource in the modern 'extended' format? (affects the interpretation of certain other `'INST'` Resource fields): 0: No – Older format 1: Yes – Current format (normal)
10	releaseLoops	Require samples used in this instrument to continue looping after note keyup? (Has no effect unless the sample resource has a defined loop): 0: No – Exit loop after keyup 1: Yes – Never exit sample loop once started
11	honorSndRate	Respect each sample resource's own sampling rate when reckoning transpositions?: 0: No – Ignore original sampling rate (not usually recommended, as this is likely to cause the instrument to play out of tune.) 1: Yes – Respect original sampling rate (This should be your default.)
12	(reserved)	Set to 0.
13	noLoops	Ignore any sample loops defined in the sample Resource(s)?: 0: No – Honor all defined loops 1: Yes – Ignore all defined loops
14	(reserved)	Set to 0.
15 (high)	(reserved)	Set to 0.

1.2.2.2. Rescaling Sample Data

It should be noted that some of the fields in a HIRF Toplevel `'INST'` Resource can be used in an alternate way to support a feature of the Headspace Audio Engine which is now considered somewhat archaic. This is a facility for altering the amplitude of the sample data in the sample resources used by the instrument, which may be desirable either for creative purposes or to pre-interpolate additional resolution; the rescaling occurs in RAM prior to the first playback use of the samples. However, this mechanism only works for sample resources with 8-bit monosample data; hence its less frequent use now that 16-bit samples are our norm.

When used, this 'rescaleSamples' facility changes the meanings of rootKey and volume fields in the `'INST'` Resource; for a single-sample instrument, this means fields 5 and 6 of the `'INST'` Resource; for a multi-sample instrument, this means the rootKey and volume fields of every keymapZone structure in the keyMap (i.e. within field 8 of the `'INST'` Resource).

To enable sample rescaling, set the INSTflags bitfield 'rescaleSamples' to 1, and set the 'miscParmsMode' bit to 0 (see section 1.2.2.1.).

Then:

- For a single-sample instrument: Fields 6 and 7 of the Toplevel `'INST'` Resource are treated not as rootKey and volume, but rather as sampleMultiplier and sampleDivisor, respectively, and every byte of the 8-bit audio sample data in the sample resource is both multiplied by the value of sampleMultiplier and divided by the value of sampleDivisor.
- Or, for a multi-sample instrument: Fields 4 and 5 of every keymapZone structure in the keyMap are treated not as rootKey and volume, but rather as sampleMultiplier and sampleDivisor, respectively, and every byte of the 8-bit audio sample data in the sample resource is both multiplied by the value of sampleMultiplier and divided by the value of sampleDivisor.

Note that most modern instruments shouldn't set either the miscParmsMode bit or the rescaleSamples bit to 0; and never for 16-bit data.

1.2.2.3. keymapZone Subresource

A keymapZone Subresource connects a contiguous range of MIDI note numbers to a specific toplevel sample resource within the current HIRF File. These keymapZone Resources are used in an instrument's keyMap (see section 1.2.2., field 8). A keyMap is a list of keymapZones, starting with the count of keymapZones in the list (expressed as a 16-bit unsigned binary integer and counting from 1 rather than 0), followed immediately by the indicated number of keymapZones, with no gaps between keymapZones.

Each keymapZone Subresource is 8 bytes long:

(Byte numbers are relative to the start of this keymapZone Subresource)

Field	Bytes	Name	Meaning and Data Format
1	1	bottomNote	Lowest MIDI note number in this zone, expressed as an 8-bit unsigned binary integer in the MIDI note number range (0 - 127), with 60 meaning 'middle C'.
2	2	topNote	Highest MIDI note number in this zone, expressed as an 8-bit unsigned binary integer in the MIDI note number range (0 - 127), with 60 meaning 'middle C'.
3	3 - 4	sampleNum	HIRF Resource ID of the `'csnd'`, `'esnd'`, or `'snd '` sample to use for this zone, expressed as a 16-bit unsigned binary integer. When converting to a 32-bit HIRF Resource ID for comparison, the top two bytes are set to 0x0000. When converting this 16-bit sample number to a 32-bit HIRF Toplevel Resource ID for purposes of searching for the sample resource, the high 2 bytes will be set to 0x0000, and the resulting Resource ID will be sought in HIRF Toplevel Resource Types `'csnd'`, `'esnd'`, and `'snd '`
4	5 - 6	rootKey[1]	Root MIDI note number to use for this sample, for this zone only, expressed as an 8-bit unsigned binary integer in the MIDI note range (0 - 127), with 60 meaning 'middle C'. Used to facilitate creative reuse of sample resources by tuning them to pitch ranges other than their original range.
5	7 - 8	volume[1]	Volume to use for this sample, for this zone only, expressed as a 16-bit unsigned binary integer, with 100 equivalent to a 'normal' volume of 100%. Used to volume-balance the keymapZones in an instrument relative to one another.

	[1] Note: When the miscParmsMode bit in the INSTflags field is set to 0, this field is interpreted differently; see **Rescaling Sample Data**, section 1.2.2.2.

1.2.2.4. The Option Subresources

Within field 13 of an `'INST'` Toplevel Resource Subbody (see section 1.2.2.1.), Option Subresources may be used to enable one or more optional features for an instrument. There are ten different Option Subresource types. All of them begin with a 4-character subresourceType field, and most of the types also have additional data fields. Some types have a fixed Subresource length, whereas others include variable-length lists among their fields; essentially, the ten types share five different formats. Each of these five formats is described in detail hereunder (sections 1.2.2.4.1 – 1.2.2.4.5); two of the formats make use of a further Subresource called an envelopeSegment, and that's described in section 1.2.2.5.

The Option Subresource types break down into three subject groups:

Basic Voice Settings:

Subresource Type	Option Name	See Section
`'ADSR'`	volEnvelope	1.2.2.4.1
`'LPGF'`	lpf	1.2.2.4.2

Voice-Based Modulation:

Subresource Type	Option Name	See Section
`'PITC'`	pitchMod	1.2.2.4.3
`'VOLU'`	volumeMod	1.2.2.4.3
`'SPAN'`	panMod	1.2.2.4.3
`'LPFR'`	lpfFreqMod	1.2.2.4.3
`'LPRE'`	lpfResonanceMod	1.2.2.4.3
`'LPAM'`	lpfMixMod	1.2.2.4.3

MIDI-Based Modulation:

Subresource Type	Option Name	See Section
`'CURV'`	performanceMod	1.2.2.4.4
`'DMOD'`	simpleWheelMod	1.2.2.4.5

1.2.2.4.1. The volEnvelope Option Subresource: `'ADSR'`

If present, this Option Subresource causes voices playing this instrument to use an amplitude envelope generator with the indicated settings:

(Byte numbers are relative to the start of this volEnvelope Option Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 4	subresourceType	The letters `'ADSR'`, expressed as four 1-byte ASCII characters.
2	5	envelopeSegsCount	Number of envelopeSegments in the following envelopeSegsList, expressed as an 8-bit unsigned binary integer, counting from 1 (not 0) and with a maximum value of 8.
3	n bytes starting at byte 6	envelopeSegsList	A list of one or more envelopeSegment structures (see section 1.2.2.5.).

1.2.2.4.2. The lpf Option Subresource: `'LPGF'`

Every voice in the HAE synthesizer passes its audio signal through a lowpass filter with parameters for cutoff frequency, resonance, and the balance of filtered vs. unfiltered signal. If present, an lpf Option Subresource statically sets the lowpass filter parameters as indicated; in the absence of an lpf Option Resource, the parameters are all set to 0.

Note: To dynamically modulate these initial parameter settings with an Envelope Generator or LFO, you must also use the voiceMod Option Subresources `'LPFR'`, `'LPRE'`, and/or `'LPAM'`.

(Byte numbers are relative to the start of this lpf Option Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 3	subresourceType	The letters `'LPGF'`, expressed as four 1-byte ASCII characters.
2	4 - 7	cornerFreq	Cutoff frequency of the lowpass filter, expressed as an unsigned 32-bit binary integer in either of two ranges, each with a different meaning: • Numbers in the range 0 - 511 map linearly to the frequencies of the fundamental pitches of MIDI note numbers 0 - 127. • Numbers in the range 512 - 16384 convert to frequencies in Hertz (F in the equation) as follows: $F = \frac{(16986 - cornerFreq) \times 22050}{16384}$ This simpler formula should however be good enough for most uses: $cornerFreq = \frac{22739 - F}{1.346}$
3	8 - 11	resonance	Resonance of the lowpass filter, expressed as an unsigned 32-bit binary integer in the range 0 - 255, with 0 meaning minimum resonance and 255 meaning maximum resonance.
4	12 - 15	mix	Balance of unfiltered (or ‘dry’) vs. lowpass-filtered output, expressed as an unsigned 32-bit binary integer in the range 0 - 255, with 0 meaning 100% unfiltered and 255 meaning 100% filtered.

1.2.2.4.3. The voiceMod Option Subresources: 'PITC', 'VOLU', 'SPAN', 'LPFR', 'LPRE', 'LPAM'

If present, each of these resources sets up one per-voice Envelope Generator and/or one per-voice Low Frequency Oscillator (LFO) as modulation sources for any one of six available destination voice parameters, as selected by the Option Subresource type:

Resource Type	Nickname	Modulates Voice Parameter
'PITC'	PitchMod	Sample Pitch
'VOLU'	VolumeMod	Volume
'SPAN'	PanMod	Pan
'LPFR'	LpfFreqMod	LPF Cutoff Frequency
'LPRE'	LpfResonMod	LPF Resonance
'LPAM'	LpfMixMod	LPF Filtered/Dry Mix

We treat these six modulation Option Subresource types as a group here because they all use the same set of fields and all work the same way. The only differences between the six destinations are a) the range of the data you should place into the fields (see table below), and b) Envelope Generators for 'VOLU' are ignored (instead, see 'ADSR' resource above).

LFO modulation is specified in terms of lfoWaveform, lfoPeriod, and lfoDepth, and the LFO is free-running; Envelope Generators are specified as a list of up to 8 envelopeSegments and an envelopeDepth, and the envelope is triggered by MIDI Note On and Note Off events.

Here are the parameter ranges and depth ranges that each destination expects to receive from its envelope generator and LFO:

Voice Parameter Modulation Ranges				
Type	**Destination**	**targetLevels**	**envelopeDepths**	**lfoDepths**
'PITC'	Pitch	-65536 - 65536[1]	-65536 - 65536[1]	-65536 - 65536[1]
'VOLU'	Volume	0 - 65536	n/a[2]	0 - 4096
'SPAN'	Pan	-65536 - 65536	-63 - 63	0 - 63[3]
'LPFR'	LPF Cutoff Frequency	0 - 65536	0 - 511, 512 - 16384[4]	0 - 8192
'LPRE'	LPF Resonance	-65536 - 65536	0 - 256	0 - 128
'LPAM'	LPF Filtered/ Dry Mix	-65536 - 65536	0 - 256	0 - 128

[1] A change of 1 in the modulator value produces 1/256 semitone of pitch deviation.

[2] `'VOLU'` does not provide an envelope– see `'ADSR'` instead (section 1.2.2.4.1.).

[3] Maximum pan position is 63 or -63.

[4] envelopeDepths 0 - 511 map linearly to frequencies of MIDI Note Numbers 0 - 127; 512 - 16384 map to frequencies as described in section 1.2.2.4.2., **The lpf Option Subresource `'LPGF'`**

Here is the data format shared by all voiceMod Option Subresources:

(Byte numbers are relative to the start of this voiceMod Option Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 3	subresourceType	The Option Subresource type, which selects the modulated voice parameter, expressed as four 1-byte ASCII characters: `'PITC'` PitchMod `'VOLU'` VolumeMod `'SPAN'` PanMod `'LPFR'` LpfFreqMod `'LPRE'` LpfResonMod `'LPAM'` LpfMixMod
2	4	envelopeSegsCount	Number of envelopeSegment Subresources in the following envelopeSegsList, expressed as an 8-bit unsigned binary integer, counting from 1 (not 0) and with a maximum of 8.
3	n bytes starting at byte 5	envelopeSegsList	A list of one or more envelopeSegment Subresources, as described in section 1.2.2.5.
4	4 bytes following field 3	lfoPeriod	Period of one LFO cycle, in microseconds, expressed as a 32-bit unsigned binary integer.

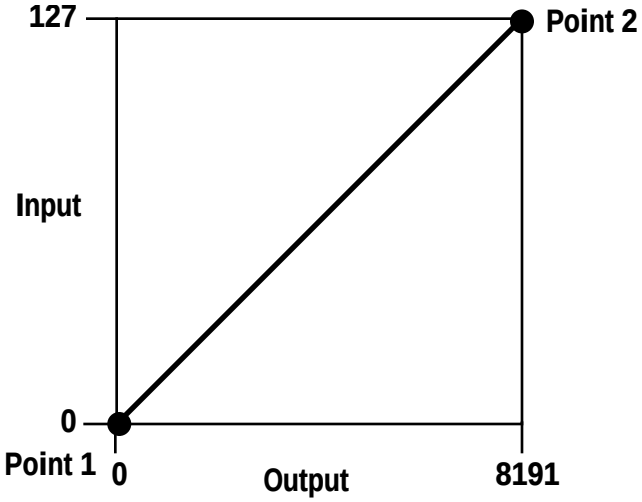
5	4 bytes following field 4	lfoWaveform	The LFO waveform, expressed as a code of four 1-byte ASCII characters: `'SINE'` Sine `'SAWT'` Sawtooth `'SAW2'` Inverted Sawtooth `'SQUA'` Square `'SQU2'` Inverted Square `'TRIA'` Triangle
6	4 bytes following field 5	envelopeDepth	Overall depth of the envelope effect, expressed as a 32-bit number; range and type of the number depends upon the subresourceType in field 1; see the **Voice Parameter Modulation Ranges** table, above.
7	4 bytes following field 6	lfoDepth	Overall depth of the LFO effect, expressed as a 32-bit number; range and type of the number depends upon the subresourceType in field 1; see the **Voice Parameter Modulation Ranges** table, above.

1.2.2.4.4. The performanceMod Option Subresource: 'CURV'

If present, these Subresources connect any one of four available MIDI performance controllers (‘source’) to any of 12 available modulatable voice parameters (‘destination’), by way of a programmable ‘transfer function’.

In general, a transfer function is a mapping from one set of numbers to another set of numbers. The transfer function available in the performanceMod Option is defined as a piecewise linear function that you define as a series of up to 8 points, as though on a graph.

In the simple case, the transfer function allows you to match the data range that the source produces to the data range the destination expects to receive. For example, the MIDI Mod Wheel produces data in the range 0 - 127; but as an instrument designer you might feel that a range of 0 - 8191 would be more useful for the effect you want to achieve with the Pitch LFO Depth destination. To create a simple linear mapping function that takes 0 - 127 as its input and produces 0 - 8191 as its output, you'd supply a list of two transferFunctionPoints: the first with a sourceLevel of 0 and a destinationLevel of 0, and the second with a sourceLevel of 127 and a destinationLevel of 8191:

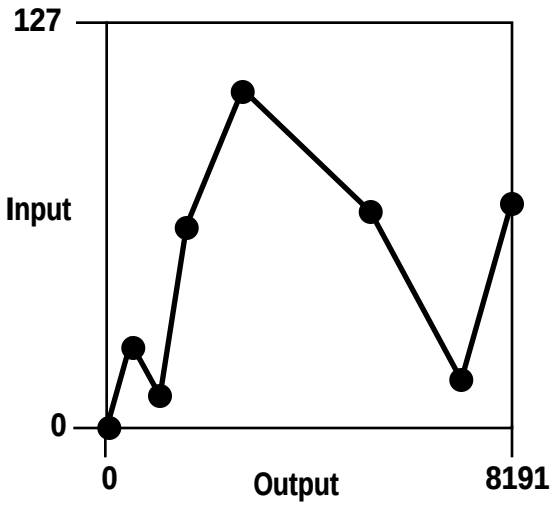


Point 1: sourceLevel 0
destinationLevel 0

Point 2: sourceLevel 127
destinationLevel 8191

By constructing more intricate transfer functions with more points, more creative effects can be

achieved; for example:



Transfer function design is expected to be a fertile area for HAE instrument designers.

(Byte numbers are relative to the start of this performanceMod Option Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 3	subresourceType	The letters `'CURV'`, expressed as four 1-byte ASCII characters.
2	4 - 7	source	The modulation source, expressed as a code of four 1-byte ASCII characters: `'PITC'` MIDI Note Number (pitch) `'VOLU'` MIDI Note Velocity `'MODW'` MIDI Mod Wheel value `'SAMP'` All instrument activity occurs within the context of a single MIDI note event. The `'SAMP'` modulation source is the number of the keymapZone into which the note event's MIDI note number falls (see field 8 of the Toplevel `'INST'` Resource Body, section 1.2.2.). This yields an integer in the range between 0 and the number of keymapZones provided for this instrument (never greater than 8).

3	8 - 11	destination	The modulation destination, expressed as a code of four 1-byte ASCII characters: **'NVOL'** Note Overall Volume **'ATIM'** Volume ADSR Attack Time **'ALEV'** Volume ADSR Attack Peak Level **'SUST'** Volume ADSR sustain level (expression controller) **'SLEV'** Volume Sustain Level **'RELS'** Volume Release Time **'PITF'** Pitch LFO Period (not rate) **'PITC'** Pitch LFO Amount **'VOLF'** Volume LFO Period **'VOLU'** Volume LFO Amount **'LPAM'** Low Pass Filter Amount **'WAVE'** Sample Fetch Pointer (offset in audio samples)
4	12	functionPointsCount	Number of functionPoint structures in field 5, expressed as an 8-bit unsigned binary integer, counting from 1 (not 0) and with a maximum value of 8.
5	n bytes starting at byte 13	functionPointsList	List of the points defining the transfer function, with each point a structure 3 bytes in length; no gaps in between structures; and a maximum of 8 points. sourceLevel: 1 byte: 8-bit unsigned binary integer destinationLevel: 2 bytes: 16-bit signed binary integer

1.2.2.4.5. The simpleWheelMod Option Subresource: `'DMOD'`

A quick-and-easy, if limited, alternative to building a more complicated performanceMod Subresource: If present, this Option Subresource sets up a sinewave LFO with a period of 0.18 seconds (rate of 5.55 Hz), connects it to the voice oscillator pitch, and makes MIDI ModWheel controller events control the depth of the effect, with full mod wheel rotation producing 1 semitone of pitch-mod depth.

Note that this Option Subresource type is unique in that it requires only the `'DMOD'` tag, and has no further fields for parameters.

(Byte numbers are relative to the start of this simpleWheelMod Option Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 3	subresourceType	The letters `'DMOD'`, expressed as four 1-byte ASCII characters.

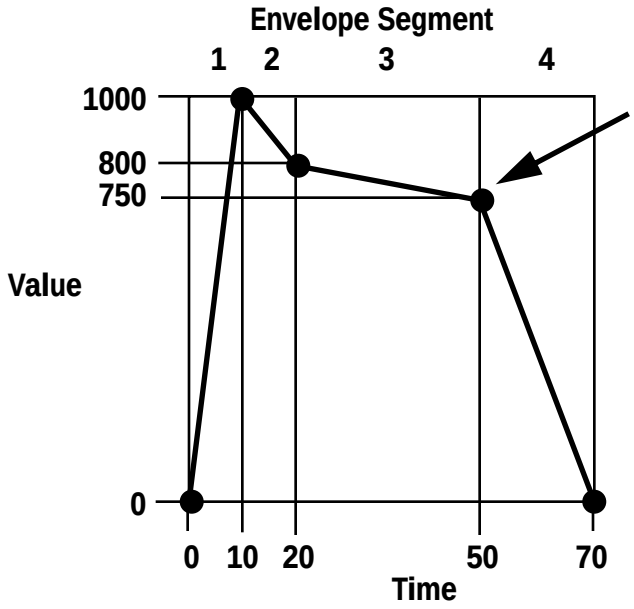
1.2.2.5. The envelopeSegment Subresource

Many of the Option Subresource types deal with musical envelope generators. The Headspace Audio Engine uses a conventional times-and-levels style of envelope, with from one to eight segments in the envelope. Each envelope segment is described with a 12-byte envelopeSegment structure. An envelope generator specification is defined as a list of envelopeSegment structures, following one another immediately with no gaps between; a 1-byte count of the number of envelopeSegments in the list must appear before the first envelopeSegment.

(Byte numbers are relative to the start of this envelopeSegment Subresource)

Field	Bytes	Name	Meaning and Data Format
1	0 - 3	targetLevel	Value to ramp towards at the end of this segment, always reaching it at endingTime, expressed as a 32-bit number whose format and range depends upon the destination parameter the envelope is intended to address; see table in section 1.2.2.4.3.
2	4 - 7	endingTime	Time in microseconds from the start of the note at which to reach the targetLevel, expressed as a 32-bit unsigned binary integer. (For segments following the `'SUST'` segment, the endingTime is treated as though the clock had stopped at the end of the `'SUST'` segment, and then restarted when the note was released.)
3	8 - 11	segmentType	The type of the segment, expressed as a code of four 1-byte ASCII characters: `'LINE'` Ordinary segment– Move the envelope generator's output value linearly from its current level towards the targetLevel, reaching targetLevel exactly at endingTime; at endingTime, move to the next segment. `'SUST'` If the MIDI note is still on when the end of this segment is reached, hover at the targetLevel until the MIDI note ends. Then, go on to the next segment. Any envelopeSegment in the list except the last one can be the `'SUST'` segment. If more than one `'SUST'` segment is present, only the last one will be treated as the `'SUST'` segment. `'LAST'` The envelope is over when the end of this segment is reached. Always end your envelope generator specification with one of these. Note that (unlike some other systems) the targetLevel for the last segment isn't required to be 0.

Example:



envelopeSegment1: 1000, 10, ‘LINE’
envelopeSegment2: 800, 20, ‘LINE’
envelopeSegment3: 750, 50, ‘SUST’
envelopeSegment4: 0, 70, ‘LAST’

1.2.3. Toplevel Resource `'snd '`

A `'snd '` Toplevel Resource represents a PCM digital audio recording (or sound 'sample'), stored as a sequence of instantaneous amplitudes, and optionally data-compressed.

In the context of sequenced MIDI music playback, a `'snd '` resource can be used either as the basis of a pitched musical instrument (by playing the same recording back at different speeds to achieve pitch transpositions), or as an audio recording for straight playback at the original pitch. In either case, an `'INST'` resource is required to link the sample in a `'snd '` resource to a MIDI Program Change number.

Alternatively, HAE also furnishes the capability to play a `'snd '` resource directly, outside of the MIDI sequencing context; see the **HAE Client API** document for details.

The Body of every HIRF Toplevel Resource with a Resource Type of `'snd '` consists of a `'snd '` Resource Subheader containing format and other non-sound information about the sample, and a `'snd '` Resource Subbody containing the sound recording (sample data). If a `'snd '` Resource Subbody is present, it follows its `'snd '` Resource Subheader immediately. The `'snd '` Resource Subheader is of variable length depending on the `'snd '` resource type, and the `'snd '` Resource Subbody is of variable length depending on the length, format, and compression characteristics of the audio sample it contains.

1.2.3.1. `'snd '` Resource Types

For compatibility with a range of authoring tools and legacy data, HAE supports three basic `'snd '` resource types, two of which have further subtypes:

- `'snd '` Type 1:
  Type 1 is designed for local file playback and includes three subtypes: Type 1 Basic, Type 1 Extended, and Type 1 Compressed.
- `'snd '` Type 2:
  Type 2 is also designed for local file playback and includes two subtypes: Type 2 Basic and Type 2 Extended.
- `'snd '` Type 3:
  Type 3 is designed for streaming compressed audio playback; for example, an internet audio feed. There are no subtypes for Type 3.

Current Headspace tools favor the following `'snd '` resource types: Type 1 Extended, Type 1 Compressed, and Type 3.

The data format for each type or subtype is different, as detailed hereunder.

1.2.3.1.1.'snd ' Resource Type 1

A Type 1 **'snd '** resource must be in one of the three defined subtype formats; a token indicating the chosen subtype must be placed in the encode field (field 14), as the interpretation of other parts of the resource data varies depending on the subtype.

(Byte numbers are relative to the start of this Toplevel **'snd '** Resource Body)

Field	Bytes	Name	Meaning and Data Format
1	0 - 1	type	Version of the **'snd '** resource format to which the resource conforms. For all Type 1 **'snd '** Resources, this is always the value 1, expressed as a 16-bit unsigned binary integer (x0001).
2	2 - 3	numDataFormats	Number of data formats present in this **'snd '** resource. For all Type 1 **'snd '** Resources, this is always the value 1, expressed as a 16-bit unsigned binary integer 0x0001).
3	4 - 5	dataFormatID	ID token for the data format to which the resource conforms. For all Type 1 **'snd '** Resources, this is always the value 5, as a token for the 'sampled sound' format, expressed as a 16-bit unsigned binary integer 0x0005).
4	6 - 9	(reserved)	For all Type 1 **'snd '** Resources, these four bytes should be set to 0x000000E0.
5	10 - 11	(reserved)	For all Type 1 **'snd '** Resources, these two bytes should be set to 0x0001).
6	12 - 13	(reserved)	For all Type 1 **'snd '** Resources, these two bytes should be set to 0x8051.
7	14 - 15	(reserved)	For all Type 1 **'snd '** Resources, these two bytes should be set to 0.
8	16 - 19	sampleHeaderOff-set	Offset from the start of the **'snd '** Resource to the start of the part of the header that deals specifically with sampled sounds (field 9). For all Type 1 **'snd '** Resources, these two bytes should be set to 20.
9	20 - 23	(reserved)	For all Type 1 **'snd '** Resources, these four bytes should be set to 0.

10	24 - 27	length *or* numChannels	For a Type 1 Basic resource, these four bytes are used to store the length in bytes of the data in the sampleArea (field 17), expressed as a 32-bit unsigned integer. For a Type 1 Extended or Type 1 Compressed resource, these four bytes are used to store the number of channels of audio data present in the audio sample data, expressed as a 32-bit unsigned binary integer. For example: for mono data, this field will be 0x0001; for stereo, it'll be 0x0002.
11	28 - 31	sampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as a 32-bit fixed-point fractional number, with the integer part in the high two bytes and the fractional part in the low two bytes. For example, a sample rate of 44,100 Hz would be expressed as 44100.0, or 0xAC440000 (As another example of the number format, the Mac OS '22k' sampling rate would be expressed as 0x56EE8BA3.)
12	32 - 35	loopStart	A Type 1 sound can have a looping section; an instrument can be set up to cause the sample to play the looping section repeatedly for as long as the note it's playing is held. For a sample with a defined loop, this field contains the location in sample frames of the first sample in the looping section of the audio sample data, expressed an a 32-bit unsigned binary integer. For a sample with no loop defined, this field should be set to 0.
13	36 - 39	loopEnd	For a sample with a defined loop (see explanation in field 12), this field contains the location in sample frames of the last sample in the looping section of the audio sample data, expressed an a 32-bit unsigned binary integer. For a sample with no loop defined, this field should be set to 0.

14	40	subtype	The subtype of this Type 1 `'snd '` resource, expressed as a 1-byte token: 0x00 Type 1 Basic 0xFF Type 1 Extended 0xFE Type 1 Compressed **Note:** The value in this field affects the interpretation of field 16 (see below).
15	41	baseNote	MIDI note number corresponding to the pitch of the recording in the audio sample data, expressed as an 8-bit unsigned binary integer in the MIDI note number range (0 - 127), with 60 corresponding to ‘middle C’.
16	For Type 1 Extended and Type 1 Compressed resources, 42 bytes of data appear here as detailed below (see sections 1.2.3.1.1.1. and 1.2.3.2.1.1.2.) and field 17 starts at byte 84; For Type 1 Basic resources, there is no field 16, and field 17 starts at byte 42.		
17	n bytes after field 16	sampleArea	The audio sample data.

1.2.3.1.1.1. Field 16 for Type 1 Extended

(Byte numbers are relative to the start of this Toplevel `'snd '` Resource Body)

Field	Bytes	Name	Meaning and Data Format
16a	42 - 45	numFrames	Length in frames of the audio sample data for this resource, expressed as a 32-bit unsigned binary integer. For mono data, one sample frame holds a single sample; for stereo data, a sample frame holds one left sample and one right sample; given the same sampling rate and duration, a mono sound and a stereo will have the same number of sample frames.
16b	46 - 55	IEEE SampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as an 80-bit MacOS floating-point number: (from the high bit to the low bit) 1 sign bit, 15 exponent bits, 1 integer bit, and 63 fraction bits.
16c	56 - 57	(reserved)	These twelve bytes should be set to zero.
16d	68 - 69	bitDepth	Number of bits per sample, expressed an a 16-bit unsigned binary integer. For example, for 8-bit data this field should contain 0x0008; for 16-bit samples, this field should contain 0x0010.
16e	70	sndIsEmbedded	Flag indicating whether or not this Toplevel `'snd '` Resource appears within an instrument bank, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: The `'snd '` **is not** in an instrument bank 1: The `'snd '` **is** in an instrument bank This should ordinarily be set to 0.
16f	71	sampleByteOrder	Flag indicating the byte order of the sample data in this resource, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: Motorola or ‘big-endian’ format 1: Intel or ‘little-endian’ format **Note:** Headspace tools favor Motorola byte order.
16g	72 - 83	(reserved)	These 12 bytes should be set to zero.

1.2.3.1.1.2. Field 16 for Type 1 Compressed

(Byte numbers are relative to the start of this Toplevel `'snd '` Resource Body)

Field	Bytes	Name	Meaning and Data Format
16a	42 - 45	numFrames	Length in frames of the audio sample data for this resource, expressed as a 32-bit unsigned binary integer. For compressed sample data, this is the compressed length; for uncompressed sample data, this is the uncompressed length. For mono data, one sample frame holds a single sample; for stereo data, a sample frame holds one left sample and one right sample; given the same sampling rate and duration, a mono sound and a stereo will have the same number of sample frames.
16b	46 - 55	IEEESampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as an 80-bit Mac OS floating-point number: bit 0 - 62: Fraction bit 63: Integer bits 64 - 78: Exponent bit 79: Sign
16c	56 - 59	(reserved)	Set these four bytes to zero always.
16d	60 - 63	compressionType	Audio data compression algorithm that produced the sample data in the sampleArea (field 17), expressed as a code of four 1-byte ACSII characters: `'none'` sample data is not compressed `'ima4'` IMA 4:1 (CCITT G.721 ADPCM) `'ulaw'` µLaw (2:1) `'alaw'` aLaw (2:1)
16e	64	force8BitSamples	Token indicating whether to override the stored bitDepth when decompressing, and instead always decompress to 8-bit samples: 0x00: Decompress to stored bit depth 0x80: Decompress to 8-bit samples

16f	65	sndIsEmbedded	Flag indicating whether or not this Toplevel `'snd '` Resource appears within an instrument bank, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: The `'snd '` **is not** in an instrument bank 1: The `'snd '` **is** in an instrument bank This should ordinarily be set to 0.
16g	66 - 67	(reserved)	These two bytes should be set to zero.
16h	68 - 71	stateVariablesPtr	This is a 4-byte temporary storage area for use by sample decompressing routines, for the purpose of saving a pointer to a state block between compression calls. You should put zeroes here, but anything you put here may be overwritten during sample decompression.
16i	72 - 75	leftoverBlockPtr	This is a 4-byte temporary storage area for use by sample decompressing routines, for the purpose of saving a pointer to any truncated samples between compression calls. You should put zeroes here, but anything you put here may be overwritten during sample decompression.
16j	76 - 77	sndIsCompressed	Flag indicating whether or not the data in the sampleArea (field 17) is compressed, expressed as a 16-bit token: 0: The sample data is uncompressed -1: The sample data is compressed Note: For compressed sample data, the compression format is indicated by the compressionType field (field 16d).
16k	78 - 81	(reserved)	These four bytes should be set to zero.
16l	82 - 83	bitDepth	Number of bits per uncompressed sample, expressed as a 16-bit unsigned binary integer. For example, for 8-bit data this field should contain 0x0008; for 16-bit samples, this field should contain 0x0010. **Note:** This field refers to the bit depth after decompressing, not the number of bits per compressed sample.

1.2.3.1.2.'snd ' Resource Type 2

(Byte numbers are relative to the start of this Toplevel 'snd ' Resource Body)

Field	Bytes	Name	Meaning and Data Format
1	0 - 1	type	Version of the 'snd ' resource format to which the resource conforms. For all Type 2 'snd ' Resources, this is always the value 2, expressed as a 16-bit unsigned binary integer (x0002).
2	2 - 3	(reserved)	For all Type 2 'snd ' Resources, these two bytes should be set to zero.
3	4 - 5	(reserved)	For all Type 2 'snd ' Resources, these two bytes should be set to 0x0001.
4	6 - 7	(reserved)	For all Type 2 'snd ' Resources, these two bytes should be set to 0x8051.
5	8 - 9	(reserved)	For all Type 2 'snd ' Resources, these two bytes should be set to 0.
6	10 - 13	sampleHeaderOff-set	Offset from the start of this 'snd ' Resource to the start of the part of this header that deals specifically with sampled sounds (field 7). For all Type 2 'snd ' Resources, these two bytes should be set to 14.
7	14 - 17	(reserved)	For all Type 2 'snd ' Resources, these four bytes should be set to 0.
8	18 - 21	length *or* numChannels	For a Type 2 Basic resource, these four bytes are used to store the length in bytes of the sample data in the sampleArea (field 15), expressed as a 32-bit unsigned integer. For a Type 2 Extended resource, these four bytes are used to store the number of channels of audio data present in the audio sample data, expressed as a 32-bit unsigned binary integer. For example: for mono data, this field will be 0x0001; for stereo, it'll be 0x0002.
9	22 - 25	sampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as a 32-bit fixed-point fractional number, with the integer part in the high two bytes and the fractional part in the low two bytes. For example, a sample rate of 44,100 Hz would be expressed as 44100.0, or 0xAC440000 (As another example of the number format, the Mac OS '22k' sampling rate would be expressed as 0x56EE8BA3.)

10	26 - 29	loopStart	A Type 2 sound can have a looping section; an instrument can be set up to cause the sample to play the looping section repeatedly for as long as the note it's playing is held. For a sample with a defined loop, this field contains the location in sample frames of the first sample in the looping section of the audio sample data, expressed an a 32-bit unsigned binary integer. For a sample with no loop defined, this field should be set to 0.
11	30 - 33	loopEnd	For a sample with a defined loop (see explanation in field 10), this field contains the location in sample frames of the last sample in the looping section of the audio sample data, expressed an a 32-bit unsigned binary integer. For a sample with no loop defined, this field should be set to 0.
12	34	subtype	The subtype of this Type 2 `'snd '` resource, expressed as a 1-byte token: 0x00: Type 2 Basic 0xFF: Type 2 Extended **Note:** The value in this field affects the interpretation of field 14 (see below).
13	35	baseNote	MIDI note number corresponding to the pitch of the recording in the audio sample data, expressed as an 8-bit unsigned binary integer in the MIDI note number range (0 - 127), with 60 corresponding to 'middle C'.
14	For Type 2 Extended resources, 42 bytes of data appear here as detailed below (see section 1.2.3.1.2.1.) and field 15 starts at byte 78; For Type 2 Basic resources, there is no field 14, and field 15 starts at byte 36.		
15	n bytes after field 14	sampleArea	The audio sample data.

1.2.3.1.2.1. Field 14 for Type 2 Extended

(Byte numbers are relative to the start of this Toplevel `'snd '` Resource Body)

Field	Bytes	Name	Meaning and Data Format
14a	42 - 45	numFrames	Length in frames of the audio sample data for this resource, expressed as a 32-bit unsigned binary integer. For mono data, one sample frame holds a single sample; for stereo data, a sample frame holds one left sample and one right sample; given the same sampling rate and duration, a mono sound and a stereo will have the same number of sample frames.
14b	46 - 55	IEEE SampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as an 80-bit MacOS floating-point number: (from the high bit to the low bit) 1 sign bit, 15 exponent bits, 1 integer bit, and 63 fraction bits.
14c	56 - 67	(reserved)	These twelve bytes should be set to 0.
14d	68 - 69	bitDepth	Number of bits per sample, expressed an a 16-bit unsigned binary integer. For example, for 8-bit data this field should contain 0x0008; for 16-bit samples, this field should contain 0x0010.
14e	70	sndIsEmbedded	Flag indicating whether or not this Toplevel `'snd '` Resource appears within an instrument bank, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0 The `'snd '` **is not** in an instrument bank 1 The `'snd '` **is** in an instrument bank This should ordinarily be set to 0.
14f	71	sampleByteOrder	Flag indicating the byte order of the sample data in this resource, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0 Motorola or ‘big-endian’ format 1 Intel or ‘little-endian’ format **Note:** Headspace tools favor Motorola byte order.
14g	72 - 83	(reserved)	These 12 bytes should be set to zero.

1.2.3.1.3.'snd ' Resource Type 3

Type 3 is Headspace's preferred format for net-streamed audio, and supports MPEG compresion standards.

(Byte numbers are relative to the start of this Toplevel **'snd '** Resource Body)

Field	Bytes	Name	Meaning and Data Format
1	0 - 1	type	Version of the **'snd '** resource format to which the resource conforms. For all Type 3 **'snd '** Resources, this is always the value 3, expressed as a 16-bit unsigned binary integer (x0003).
2	2 - 5	compressionType	Audio data compression algorithm that produced the sample data in the sampleArea (field 19), expressed as a code of four 1-byte ACSII characters: **'none'** sample data is not compressed **'ima4'** IMA 4:1 (CCITT G.721 ADPCM) **'ulaw'** µLaw (2:1) **'alaw'** aLaw (2:1) **'mpga'** Headspace mpeg, 40k bits per second **'mpgb'** Headspace mpeg, 48k bits per second **'mpgc'** Headspace mpeg, 56k bits per second **'mpgd'** Headspace mpeg, 64k bits per second **'mpge'** Headspace mpeg, 80k bits per second **'mpgf'** Headspace mpeg, 96k bits per second **'mpgg'** Headspace mpeg, 112k bits per second **'mpgh'** Headspace mpeg, 128k bits per second **'mpgi'** Headspace mpeg, 160k bits per second **'mpgj'** Headspace mpeg, 192k bits per second **'mpgk'** Headspace mpeg, 224k bits per second **'mpgl'** Headspace mpeg, 256k bits per second **'mpgm'** Headspace mpeg, 320k bits per second **'mpgn'** Headspace mpeg, 32k bits per second

3	6 - 9	sampleRate	Sampling rate at which the audio sample data was created, in Hertz, expressed as a 32-bit fixed-point fractional number, with the integer part in the high two bytes and the fractional part in the low two bytes. For example, a sample rate of 44,100 Hz would be expressed as 44100.0, or 0xAC440000 (As another example of the number format, the Mac OS '22k' sampling rate would be expressed as 0x56EE8BA3.)
4	10 - 13	uncompressedLength	Number of bytes of audio sample data when uncompressed, expressed as a 32-bit unsigned binary integer. Note that for all compressionTypes except 'none', this number should be different from the lengthInBytes (field 4).
5	14 - 17	numFrames	Number of audio sample frames in the sample data, expressed as a 32-bit unsigned binary integer. For mono data, one sample frame holds a single sample; for stereo data, a sample frame holds one left sample and one right sample; given the same sampling rate and duration, a mono sound and a stereo will have the same number of sample frames.
6	18 - 21	lengthInBytes	Number of bytes of data in the sampleArea (field 20), expressed as a 32-bit unsigned binary integer. Note that for all compressionTypes except 'none', this number should be different from the uncompressedLength (field 4).
7	22 - 25	bytesPerFrame	Size in bytes of every sample frame in the sampleArea, expressed as a 32-bit unsigned binary integer. (Not used for compressionType 'none'.)
8	26 - 29	pcmDataOffset	Number of uncompressed samples to trim off the start of the sound (to compensate for latency in audio compressors such as MPEG), expressed as a 32-bit unsigned binary integer. Note that to save memory, HAE will not store the trimmed samples in the decompression buffers.

9	30 - 53	loopStartArray	A Type 3 sound can loop. These fields contain an array of six loopStarts and an array of six loopEnds, with each slot containing the sample frame numbers of the first frame and last frame of the loop for each of the up to six audio channels in this Toplevel `'snd '` resource; the frame numbers are expressed as 32-bit unsigned binary integers. Loop bounds for unused channels should be set to 0. **Note:** Providing separate loop bounds for each channel permits the channels to have different sampling rates, and offers new creative options.
10	54 - 77	loopEndArray	
11	78 - 81	nameResource-Type	To link a text annotation to this Toplevel `'snd '` resource, put the annotation into a Toplevel Resource in the same HIRF File, put that resource's 4-character Resource Type code into field 11, and put that resource's 32-bit Resource ID number into field 12. If you don't have an annotation for this Toplevel `'snd '` resource, set field 11 to zeroes; then field 12 will be ignored.
12	82 - 85	nameResourceID	
13	86	baseNote	MIDI note number corresponding to the pitch of the recording in the audio sample data, expressed as an 8-bit unsigned binary integer in the MIDI note number range (0 - 127), with 60 corresponding to 'middle C'.
14	87	channels	Number of audio channels in the sample data, expressed as an 8-bit unsigned binary integer. For mono sounds this will be 1; 2 for stereo; or 6 for 5.1-channel audio.
15	88	bitDepth	Number of bits per uncompressed sample, expressed as an 8-bit unsigned binary integer. For 16-bit audio, this will be 16; for 8-bit, this will be 8.
16	89	sndIsEmbedded	Flag indicating whether or not this Toplevel `'snd '` Resource appears within an instrument bank, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: The `'snd '` **is not** in an instrument bank 1: The `'snd '` **is** in an instrument bank This should ordinarily be set to 0.

17	90	sndIsEncrypted	Flag indicating whether or not the data in the sampleArea (field 20) is encrypted, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: The sample data **is not** encrypted 1: The sample data **is** encrypted This should ordinarily be set to 0. Note that the sample data may be both data-compressed and encrypted.
18	91	sampleByteOrder	Flag indicating the byte order of the uncompressed sample data, expressed as a 1-bit binary flag stored in the low bit of a single byte: 0: The sample data is in Motorola (“big-endian’) format 1: The sample data is in Intel (“little-endian’) format
19	92 - 125	(reserved)	These 34 bytes are reserved and should be set to 0.
20	n bytes starting at byte 126	sampleArea	The audio sample data, which may be compressed and/or encrypted, appears here.

1.2.4. Toplevel Resource `'VERS'`

The Toplevel Resource supports a standard 3-number version tracking scheme (i.e. “8.1.1” as in an application release designation) for use in version-tracking media files. This is a simple 6-byte resource with three fields.

(Byte numbers are relative to the start of this Toplevel `'VERS'` Resource Body)

Field	Bytes	Name	Meaning and Data Format
1	0 - 1	firstNumber	First number, expressed as a 16-bit unsigned binary integer. This is often called the ‘major release’.
2	2 - 3	secondNumber	Second number, expressed as a 16-bit unsigned binary integer. This is often called the ‘minor release’.
3	4 - 5	thirdNumber	Third number, expressed as a 16-bit unsigned binary integer. This is often called the ‘fix release’.

2. An Example RMF File

This section presents an example RMF File in diagrammed 'pseudo-code' that a programmer can understand quickly. This RMF File consists of three HIRF Resources: an **'ecmi'**, a **'SONG'**, and a **'CACH'**.

```
//-------------------------------------------------------------------------------
//   "Disassembly" of RMF file 'modern-rock.rmf'
//-------------------------------------------------------------------------------

[ Offsets are in hex and are relative to the start of the file ]

     HIRF File Header ---------------------------------------------------------
     |
0000:|    'I' 'R' 'E' 'Z'       // File master chunk type
0004:|    0x00000001            // File conforms to version 1 of the HIRF format
0008:|    0x00000003            // 3 HIRF Resources present in this file
     |
000B:----End of HIRF File Header ------------------------------------------------

000C:First HIRF Resource -------------------------------------------------------
     |
     |    HIRF Resource Header --------------------------------------------------
     |    |
     |    |    0x00002E30       // Next HIRF Resource starts at
     |    |                     //   file offset 0x2E30
0010:|    |    'e' 'c' 'm' 'i'  // This HIRF Resource is of type 'ecmi'
0014:|    |    0x00003530       // This is HIRF 'ecmi' Resource no. 0x3530
0018:|    |    0x0b             // Name is 11 bytes long
     |    |    "modern rock"    // The Name of this HIRF 'ecmi' Resource is
     |    |                     //   "modern rock" (Pascal string, no final NULL)
0024:|    |    0x00002E08       // The Body of this HIRF 'ecmi' Resource
     |    |                     //   is 0x2E08 bytes long
     |    |
0027:|    ----End of HIRF Resource Header ----------------------------------------
     |
0028:|    HIRF 'ecmi' Resource Body ---------------------------------------------
     |    |
     |    |    0x2E30 bytes of 'ecmi'-format data here:
     |    |
     |    |                     // 'ecmi' data is an encrypted and data-
     |    |                     //   compressed Standard MIDI File.
     |    |                     //   'ecmi' is one of RMF's formats for
     |    |                     //   storing a musical composition
     |    |                     //   and/or performance.
     |    |
2E2F:|    ----End of HIRF 'ecmi' Resource Body -----------------------------------
     |
     ----End of First HIRF Resource ------------------------------------------------

2E30:Second HIRF Resource ------------------------------------------------------
```

```
      |
      |     HIRF Resource Header ------------------------------------------------
      |     |
      |     |     0x00002FF5           // Next HIRF Resource starts at
      |     |                          //   file offset 0x00002FF5
2E34: |     |     'S' 'O' 'N' 'G'      // This HIRF Resource is of type 'SONG'
2E38: |     |     0x0000680C           // This is HIRF 'SONG' resource no. 0x680C
2E3C: |     |     0x0f                 // Name is 15 characters long
2E3D: |     |     "Modern-Rock.rmf"    // The name of this HIRF 'SONG' Resource is
      |     |                          //   "Modern-Rock.rmf" (Pascal string)
2E4C: |     |     0x000001A5           // The body of this HIRF 'SONG' Resource
      |     |                          //   is 0x01A5 bytes long
      |     |
2E4F: |     ----End of HIRF Resource Header ----------------------------------------
      |
2E50: |     HIRF 'SONG' Resource Body ------------------------------------------
      |     |
      |     |     0x01A5 bytes of 'SONG'-format data here:
      |     |
2E50: |     |     'SONG' Resource Subheader ------------------------------------
      |     |     |
      |     |     |     0x3530               // Use music data in RMF Resource ID number
      |     |     |                          //   0x3530 to play this 'SONG'
2E52: |     |     |     0x00                 // Reserved byte
2E53: |     |     |     0x04                 // Use reverb type 4 (== 'Lab')
      |     |     |                          //   when playing this song
2E54: |     |     |     0x0000               // Use tempo stored in the music data
2E56: |     |     |     0x01                 // Music data for this 'SONG' is in RMF
      |     |     |                          //   structured format]
2E57: |     |     |     0x01                 // Resources for this 'SONG' are encrypted
      |     |     |                          //    [1 == encryted, 0 == not]
2E58: |     |     |     0x0000               // Don't offset MIDI Note numbers (pitches)
2E5A: |     |     |     0x0004               // Max of 4 digital audio file
      |     |     |                          //  playback voices at once
2E5C: |     |     |     0x001C               // Max of 28 music synth voices at once
2E5E: |     |     |     0x0008               // Scale levels for up to 8 full-scale
      |     |     |                          //   signals at once
2E60: |     |     |     0x0000               // Use volume stored in the music data
2E62: |     |     |     0x00                 // This 'SONG' not embedded in an instrument
      |     |     |                          //   bank (1==embedded, 0==not)
2E63: |     |     |     0x00                 // Reserved byte
2E64: |     |     |                          // 7 longs of unused/reserved data here
2E68: |     |     |     0x00000000 0x00000000 0x00000000 0x00000000
2E6C: |     |     |     0x00000000 0x00000000 0x00000000
2E80: |     |     |     0x000B               // 11 Song Resources follow immediately,
      |     |     |                          //   in the Body section
      |     |     |
2E81: |     |     ----End of 'SONG' Resource Subheader -----------------------------
      |     |
2E82: |     |     'SONG' Resource Subbody --------------------------------------
      |     |     |
      |     |     |     List of 11 Song Resources here:
      |     |     |     (each consisting of a 4-byte type selector header
      |     |     |      plus a variable-length body; all text subresources,
      |     |     |      all encoded.)
      |     |     |
      |     |     |     Song Subresource 1-------------------------------------------
      |     |     |     |     // 'TITL': Song Title text resource
      |     |     |     |     'T' 'I' 'T' 'L'
```

```
      |     |     |     |    // encrypted string ("modern rock")
2E86: |     |     |     |    0x91 F8 FB 32 15 9D FA 4A 28 A4 16 64
2E91: |     |     |     ----End of Song Subresource 1 ------------------------------
      |     |     |
2E92: |     |     |     Song Subresource 2-----------------------------------------
      |     |     |     |    // 'COMP': Composer Name text resource
      |     |     |     |    'C' 'O' 'M' 'P'
      |     |     |     |    // encrypted string
2E96: |     |     |     |    0x94 66 D5 3D 4B 8F D5 CA 1D 82
2E9F: |     |     |     ----End of Song Subresource 2 ------------------------------
      |     |     |
2EA0: |     |     |     Song Subresource 3-----------------------------------------
      |     |     |     |    // 'COPD': Copyright Date text resource
      |     |     |     |    'C' 'O' 'P' 'D'
      |     |     |          // encrypted string
2EA4: |     |     |     |    0x75 2C 4C A2 D2 79 6B 5C 98 81 39 5F 5E 97 43 95
      |     |     |     |    0xC0 DC 8A 6A 05 8C 60 CC D7 69 78 F4 B7 4B 16 A4
      |     |     |     |    0x6D F6 D4 6E
2EC7: |     |     |     ----End of Song Subresource 3 ------------------------------
      |     |     |
2EC8: |     |     |     Song Subresource 4-----------------------------------------
      |     |     |     |    // 'LICC': Licensing Contact text resource
      |     |     |     |    'L' 'I' 'C' 'C')
      |     |     |     |    // encrypted string
2ECC: |     |     |     |    0xB4 A4 8B DA FF 38 32 5D 2F 6B 11 80 BC 68 D8 B7
      |     |     |     |    0x65 1E EA 10 32 62 22 69 18 BD 12 D7 08 0E DF EE
      |     |     |     |    0xB2 1F 1E BD CF
2EF0: |     |     |     ----End of Song Subresource 4 ------------------------------
      |     |     |
2EF1: |     |     |     Song Subresource 5-----------------------------------------
      |     |     |     |    // 'LUSE': Use of License text resource
      |     |     |     |    'L' 'U' 'S' 'E'
      |     |     |     |    // encrypted string
2EF5: |     |     |     |    0x8E 61 4B 82 06 72 F5 30 C6 3B 9E 39 AE FA F3 C0
      |     |     |     |    0x54 C4 C0 2C 5E 15 21 24 88 2C F4 B9 33 3A 54 83
      |     |     |     |    0xEE 70 4A 08 64 95 65 AE
2F1C: |     |     |     ----End of Song Subresource 5 ------------------------------
      |     |     |
2F1D: |     |     |     Song Subresource 6-----------------------------------------
      |     |     |     |    // 'LDOM': Licensed Domain text resource
      |     |     |     |    'L' 'D' 'O' 'M'
      |     |     |     |    // encrypted string
2F21: |     |     |     |    0xF1 2D D6
2F23: |     |     |     ----End of Song Subresource 6 ------------------------------
      |     |     |
2F24: |     |     |     Song Subresource 7-----------------------------------------
      |     |     |     |    // 'LTRM': Term of License text resource
      |     |     |     |    'L' 'T' 'R' 'M'
      |     |     |     |    // encrypted string
2F28: |     |     |     |    0xED E7 1E D6 9A 49 D8
2F2E: |     |     |     ----End of Song Subresource 7 ------------------------------
      |     |     |
2F2F: |     |     |     Song Subresource 8-----------------------------------------
      |     |     |     |
      |     |     |     |    // 'EXPD': License Expir. Date text resource
      |     |     |     |    'E' 'X' 'P' 'D'
      |     |     |     |    // encrypted string
2F33: |     |     |     |    0xED E7 1E D6 9A 49 F8 85 32 7D 71 8C A9 A0 DB 90
      |     |     |     |    0x31 4F 74 03 D9 7A 66 17 34 C2 DD 41 23 1C D7 2C
```

```
      |     |     |     |    0xC1
2F53: |     |     |     ----End of Song Subresource 8 ------------------------------
      |     |     |
2F54: |     |     |     Song Subresource 9-----------------------------------------
      |     |     |     |    // 'NOTE': General-purpose Note text resource
      |     |     |     |    'N' 'O' 'T' 'E'
      |     |     |     |    // encrypted string
2F58: |     |     |     |    0x9D 2B AE 16 C2 8E 6E F0 72 4A F4 A7 3C 3C C8 4B
      |     |     |     |    0xAD C1 13 D0 98 23 74 85 D7 E0 FE AB 61 A5 8E D8
      |     |     |     |    0x4C 6C FE CD A7 28 33 BA 49 35 CC 98 BA B4 90 88
      |     |     |     |    0x57 76 A5 7F 14 CB 03 95 EF C2 78 58 36 C3 C4 48
      |     |     |     |    0xC1 B7 40 F8 50 C2 3C 4F EC BC 14 7D B5 05 57 89
      |     |     |     |    0x51 EC 89 F1 95 51 99 77 58 8C 7C 85 A9 C5 63 E3
      |     |     |     |    0x00 02 BE 6A 74 1B 7F 4D 53 71 22 7C 08 BB 35 AB
      |     |     |     |    0x8C C6 F1 D2 B6 EE 57 81 19 66 BD CB 07 5A 4B D3
      |     |     |     |    0xF8 D3 28 2B E9 01 9E ED
2FDF: |     |     |     ----End of Song Subresource 9 ------------------------------
      |     |     |
2FE0: |     |     |     Song Subresource 10----------------------------------------
      |     |     |     |    // 'INDX': Index Number text resource
      |     |     |     |    'I' 'N' 'D' 'X'
      |     |     |     |    // encrypted string
2FE4: |     |     |     |    0xF1 2D D6
2FE6: |     |     |     ----End of Song Subresource 10 -----------------------------
      |     |     |
2FE7: |     |     |     Song Subresource 11----------------------------------------
      |     |     |     |    // 'PERF': Performed By text resource
      |     |     |     |    'P' 'E' 'R' 'F'
      |     |     |     |    // encrypted string
2FEB: |     |     |     |    0x94 66 D5 3D 4B 8F D5 CA 1D 82
2FF4: |     |     |     ----End of Song Subresource 11 -----------------------------
      |     |     |
      |     |     ----End of 'SONG' resource subbody ------------------------------
      |     |
      |     ----End of HIRF 'SONG' Resource Body -----------------------------------
      |
      ----End of Second HIRF Resource ------------------------------------------

2FF5: Third HIRF Resource -------------------------------------------------------
      |
      |     HIRF Resource Header --------------------------------------------------
      |     |
      |     |     0x00003046      // Next HIRF Resource would start at file offset
      |     |                     //   0x3046, if there were one. Since this is the
      |     |                     //   last HIRF Resource, this is the offset of the
      |     |                     //   end-of-file + 1.
2FF9: |     |     'C' 'A' 'C' 'H' // This HIRF Resource is of type 'CACH'
2FFD: |     |     0x00000000      // This is HIRF 'CACH' Resource no. 0
3001: |     |     0x00            // This 'CACH' resource is unnamed
      |     |                     //   (null P-string == just 0x00, no characters)
3002: |     |     0x00000040      // The Body of this 'CACH' resource is
      |     |                     //   0x40 bytes long
      |     |
3005: |     ----End of HIRF Resource Header ----------------------------------------
      |
3006: |     HIRF Resource Body ----------------------------------------------------
      |     |
      |     |     0x40 bytes of 'CACH'-format data here:
      |     |
```

```
      |    |                   // An optional resource type used for speeding
      |    |                   //   up access to other HIRF Resources by
      |    |                   //   storing their names and start offsets here
      |    |                   //   in a small index database.  Without a 'CACH'
      |    |                   //   entry, the only way to access any particular
      |    |                   //   HIRF file resource is to parse the file /
      |    |                   //   walk the structure.
      |    |
3045: |    ----End of HIRF Resource Body ------------------------------------------
      |
      ----End of Third HIRF Resource ------------------------------------------------

3045: Last byte of Third HIRF Resource data, and
      End Of File ------------------------------------------------------------------
```

Index