

Apple II File Type Notes



Developer Technical Support

File Type: **\$D8 (216)**
Auxiliary Type: **\$0002**

Full Name: Apple IIGS Sampled Instrument File
Short Name: ASIF File

Written by: John Worthington & Matt Deatherage

March 1989

Files of this type and auxiliary type contain instruments in the Apple IIGS Sampled Instrument Format (ASIF).

The Apple IIGS Sampled Instrument Format (ASIF) is the standard format used for storing Apple IIGS Sampled Instruments on disk. All sampled instruments supplied by Apple for the Apple IIGS Note Synthesizer will use this format. Likewise, all utilities supporting Note Synthesizer sampled instruments will also use this format.

ASIF is designed around the needs of the current Apple IIGS Note Synthesizer and Apple IIGS sound hardware. While the format of ASIF (especially `INST` chunks) is greatly influenced by the Note Synthesizer, the information may be sufficient for other sampled sound synthesizers to accurately render the sound.

Most instrument files for the Apple IIGS have a ProDOS file type of \$D6. ASIF files are instead identified as file type \$D8, auxiliary type \$0002 because of their sampled nature. All other instruments for the the Apple IIGS will be identified by file type \$D6.

Note: Earlier ASIF documentation, not widely circulated, defined ASIF files as file type \$CA, auxiliary type \$8000. As documented in this Note, ASIF has been reassigned to file type \$D8 and auxiliary type \$0002. Applications which read files based on the old file type and auxiliary type should not perform adversely, since no other application should be creating files with that combination. However, we strongly urge developers to create ASIF files with file type \$D8 and auxiliary type \$0002 only. We further encourage developers to revise existing programs to use this new assignment at their earliest convenience.

ASIF files conform to the “EA IFF 85” Standard for Interchange Format Files developed by Electronic Arts. Electronic Arts additionally has some public domain code available for reading and writing IFF files.

ASIF is provided more for compatibility than for interchange. It is a highly Apple IIGS specific file format. Those wishing the highest level of sampled sound compatibility across programs

and CPUs should use Audio Interchange File Format (Audio IFF). Audio IFF is documented in Apple II File Type Note for file type \$D8, auxiliary type \$0000.

This Note defines the required chunks **INST** and **WAVE**, as well as the optional (“**NAME**”), copyright (“(c)”), author (“**AUTH**”), and annotation (“**ANNO**”) chunks. These are all “standard” chunks. Additional chunks for private or future needs may be added later. Figure 1, located at the end of this Note, illustrates the ASIF format in a box diagram.

Required Data Chunks

An ASIF file consists of a single **FORM** ASIF, which contains one and only one **WAVE** chunk and one or more **INST** chunks. Each ASIF file defines at least one instrument.

INST chunks contain all of the Note Synthesizer specific information needed to define an instrument, exclusive of the actual wave form. The information in the **INST** chunk defines the characteristics of an instrument such as the envelope, pitch range, and maximum pitch bend. There must be at least one **INST** chunk for each instrument in the ASIF file.

WAVE chunks contain the waveforms for a given instrument. A **WAVE** chunk may contain waveforms used by more than one instrument. In most cases, the waveforms used by an application will be merged into a single 64K block that is loaded into DOC RAM when the application is launched. In this case, there would be several **INST** chunks referring to that single **WAVE** chunk. Most music applications will probably store instruments one to a file, which is the preferred way of distributing ASIF instruments.

Note: The length of any chunk must be even. If a chunk has an odd length, a pad byte of \$00 must be added to the end of the chunk. The pad byte, if present, should never have a value other than \$00.

The FORM Chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be “FORM.”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and ckID. You may think of this value as the offset to the end of the chunk. Note that this is a Reverse Long; the bytes are stored high byte first.
ckType (chunks...)	4 Types	The type of chunk. These four bytes must be “ASIF.”

Immediately following the 12-byte **FORM** chunk header are the data chunks of the ASIF file. There must be one and only one **WAVE** chunk, and at least one **INST** chunk. Optionally there may be name (“**NAME**”), copyright (“(c)”), author (“**AUTH**”), or annotation (“**ANNO**”) chunks. All data chunks are part of the larger **FORM** chunk, referred to as the **FORM ASIF** because of the ID and Type of this chunk.

The INST Chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be "INST."
ckSize	Rev. Long	The length of this chunk, excluding ckSize and ckID. You may think of this value as the offset to the end of the chunk. Note that this is a Reverse Long; the bytes are stored high byte first.
InstName	String	A Pascal String containing the name of the instrument referred to by this INST block. This string should be used as the display name of the instrument.

Note: The length byte of InstName is also referred to as INameLength.

SampleNum	Word	The number of the sample in the WAVE chunk to which this instrument refers.
Envelope	8 InstSegs	Eight linear InstSegs defining the instrument's envelope.

The InstSeg is a three-byte linear segment that describes a level and a slope. The level is called the breakpoint and represents the linear amplitude of the sound. The slope is described by an increment added or subtracted from the current level at the update rate. Regardless of the increment, the breakpoint will never be exceeded. All ASIF instruments assume an update rate of 200 Hz. The increment is a two-byte fixed pointer; that is, the lower eight bits represent a fraction. Thus when the increment is one, it represents 1/256. In this case, the increment would have to be added 256 times (1.28 seconds) to cause the level to go up by 1. At a 200 Hz update rate each increment takes 5 milliseconds. If an application wishes to use an update rate other than 200 Hz, the envelope must be scaled as necessary. If the envelope is not scaled, the instrument will not sound correct.

The breakpoint is a byte between 0 and 127 (\$00 and \$7F). It should represent sound level in a logarithmic scale: every 16 steps change the amplitude by 6 dB.

Therefore the envelope is composed of eight InstSegs:

stage1	Byte	Breakpoint 1
	2 Bytes	Increment 1
stage2	Byte	Breakpoint 2
	2 Bytes	Increment 2
stage3	Byte	Breakpoint 3
	2 Bytes	Increment 3
stage4	Byte	Breakpoint 4
	2 Bytes	Increment 4
stage5	Byte	Breakpoint 5
	2 Bytes	Increment 5
stage6	Byte	Breakpoint 6
	2 Bytes	Increment 6
stage7	Byte	Breakpoint 7
	2 Bytes	Increment 7
stage8	Byte	Breakpoint 8
	2 Bytes	Increment 8

Increment 1 is used to go from the initial level of 0 up to the level of breakpoint 1. Increment 2 is used to go from breakpoint 1 to breakpoint 2, and so on. The sustain level of the envelope, if there is one, is created by setting the increment to zero, causing the envelope to get stuck on that level. The last segment used for release should always have a breakpoint of zero, so the sound eventually reaches silence. Unused segments should have a zero breakpoint and a non-zero increment.

ReleaseSegment	Byte	Specifies the release segment of the envelope. This must be a number from 1 to 7. The release may take several segments to get to zero. The last segment should always be zero.
PriorityIncrement	Byte	A number that will be subtracted from the generator priority when the envelope reaches the sustain segment. The sustain segment is the first segment with a zero increment. When the release segment is reached, the priority is cut in half. The priority of each generator is also decremented by one each time a new generator is allocated. This causes older notes to be preferred for stealing.
PitchBendRange	Byte	The number of semitones that the pitch will be raised when the pitchwheel reaches 127 (the center value is 64). The legal values for PitchBendRange are 1, 2, and 4.
VibratoDepth	Byte	The initial fixed depth of vibrato, ranging from 0 to 127. Vibrato is a triangular-shaped Low Frequency Oscillator (LFO) modulating the pitch of both oscillators in a generator. A VibratoDepth of zero turns the vibrator mechanism off, which saves some CPU time (since vibrato is implemented in software).
VibratoSpeed	Byte	Controls the rate of the vibrato LFO. It can be any byte value, although the range from 5 to 20 is most useful. The frequency range is linear, in 0.2 Hz steps.
UpdateRate	Byte	Unused; set to zero. Previous versions of ASIF listed this byte as the update rate in .4 Hz, but a one-byte field is not large enough to provide suitable resolution (102 Hz is the maximum allowed), much less the standard Note Synthesizer value of 200 Hz (the byte would have to hold the value 500; not an easy task for a byte). All ASIF instruments are assumed to have an update rate of 200 Hz.
AWaveCount	Byte	The number of waves in the following AWaveList. There can be up to 255 waves in the AWaveList.
BWaveCount	Byte	The number of waves in the following BWaveList. There can be up to 255 waves in the BWaveList.
AWaveList	AWaveCount	Waves.
BWaveList	BWaveCount	Waves.

The WaveList structure is a variable-length array where each entry is six bytes long. The information is particular to the DOC, and the developer should refer to the DOC information in the *Apple IIGS Hardware Reference* and the *Apple IIGS Toolbox Reference Update* when creating instruments. Each six-byte entry represents a waveform and contains information about the allowable pitch range of the waveform. This means that the waves can be “multi-sampled” across an imaginary keyboard. When a note is played, WaveListA and WaveListB will be examined, and one waveform will be picked and assigned to each oscillator.

Each wave in a WaveList has the following 6-byte format:

TopKey	Byte	The highest MIDI semitone this waveform will play. The Note Synthesizer will examine the TopKey field of each waveform until it finds one greater than or equal to the note it is trying to play. The items in the WaveList should be in order of increasing TopKey values. The last wave should have a TopKey value of 127. The TopKey value is the split point between the waveforms.
--------	------	---

The next three bytes will be stuffed into the DOC registers:

WaveAddress	Byte	The high byte of the waveform address. Note that the value selected for WaveAddress should assume that the waveform starts in page zero. When the waveform is actually placed in DOC RAM, the values must be adjusted as appropriate. As an example, for a waveform starting at \$8000 in DOC RAM, this value would be \$80.
WaveSize	Byte	Sets both the size of the wavetable and the frequency resolution.
DOCMode	Byte	Placed in the DOCs Mode register. The interrupt-enable should always be zero.

Some ways this may be used are:

Synthesizer (\$00), where both oscillators (A and B) run in free run mode

Sample, no loop: Oscillator A in swap mode (\$06) and oscillator B in one-shot halted mode (\$03). Oscillator A will play its wave once and start Oscillator B, which will play its wave to the end once and stop.

Sampled with loop: Oscillator A in swap mode (\$06), and Oscillator B in free-run halted mode (\$01). Oscillator A will play its wave once and then start Oscillator B, which will play continuously until the note ends.

The high nibble of the DOCMode is the channel number. This must be set correctly for stereo output. While all of the currently available stereo cards will map even-numbered channels to the right and odd-numbered channels to the left, software should use channel 0 for right and channel 1 for left. This will ensure compatibility with cards that provide more than two channels of output. If you are not designing stereo instruments, always set the channel to zero.

RelPitch	Word	Used to tune the waveform. This will compensate for different sample rates and waveform sizes. The high byte is in semitones, but can be a signed number. The low byte is in 1/256 semitone increments. Note that the low byte is first in memory; this is a regular 65816 Word. A setting of zero is the default for sounds that gave one cycle per page of waveform memory.
----------	------	---

The WaveList structure is designed to give greater flexibility in creating realistic instrumental timbres. It allows “multi-sampling” with different samples of sounds on different ranges of pitch. It allows mixing of various sized wave forms, with different tuning on each semitone, to allow separate tuning of each note. This is one way to

duplicate special tuning systems like “just temperament.” The wave pointers need not be different in this case, just the RelPitch fields.

Tuning is accurate to 1/128 of a semitone in the Note Synthesizer, subject to the resolution setting of the DOC. For accurate tuning on lower notes, it may be necessary to use higher settings in the DOC resolution register.

Note: The Audio Interchange File Format (Audio IFF) also has a chunk named “INST” which will appear to a standard IFF reader the same as the ASIF “INST” chunk. To tell the two apart, check the ckSize field. The Audio IFF “INST” chunk will **always** have ckSize of 20 bytes, and the ASIF “INST” chunk will **never** have a chunk size of 20 bytes.

The WAVE chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be “WAVE.”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and ckID. You may think of this value as the offset to the end of the chunk. Note that this is a Reverse Long; the bytes are stored high byte first.
WaveName	String	A Pascal String containing the name of the waveform referred to by this WAVE block.

Note: The length byte of WaveName is also referred to as WaveNameLen.

WaveSize	Word	The size of the waveform WaveData, in bytes. WaveSize may be any value from \$0000 to \$FFFF. This is a zero-based counter; WaveData that is one byte long would result in a WaveSize of \$0000. This allows full 64K WaveData entries.
NumSamples	Word	The number of different sounds in this WAVE chunk. NumSamples describes the number of entries in SampleTable. Note that this is not necessarily the number of instruments. Although not required, there should be a WaveList entry in an INST chunk for each entry in the SampleTable.
SampleTable	NumSamples	Samples.

SampleTable is a table of the waveforms corresponding to different “samples”. Each entry in SampleTable is 12 bytes long. Each sample entry is defined as follows:

Location	Word	The byte offset to the waveform from the beginning of the WAVE chunk.
Size	Word	The size of the waveform in 256-byte pages. Size is specified in pages since the sample size passed to the DOC must be in pages.
OrigFreq	Fixed	The original frequency that was sampled, in hertz. For example, if A440 was sampled, the value of this field would be 440.00. A value of zero in this field means that the original frequency of the sample is unknown.
SampRate	Fixed	The sample rate used to generate this sample, in hertz. A value of zero in this field means that the original sample rate is unknown.

There are NumSamples of these sample entries in the SampleTable.

WaveData

WaveSize Bytes The actual waveform. The DOC uses samples in an eight-bit linear format. A value of \$80 is considered to be a zero crossing. Positive values are greater than \$80; negative values are less than \$80. Although WaveData may contain zeros as oscillator control values, it should **never** contain a zero value as a sample value since this halts the oscillator.

Optional Data Chunks

There are currently three types of optional data chunks. These chunks may be included in an ASIF file if desired. They are considered part of the set of “standard” chunks in the Electronic Arts “*EA IFF 85*” definition.

The NAME Chunk

This chunk names the instrument of collection of instruments defined in the ASIF file. This chunk may be used to supply a display name for a collection of instruments. This can be useful since IFF programs know about the NAME chunk, but may not know about the name field in INST or WAVE chunks.

ckID	4 Bytes	The ID for this chunk. These four bytes must be “NAME.”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and cdID.
Name	Bytes	ASCII characters (\$20–\$7F) representing the name. There should be ckSize characters.

The AUTH Chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be “AUTH.”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and cdID.
author	Bytes	ASCII characters (\$20–\$7F) representing the name of the author of the voices or collection of voices. There should be ckSize characters.

The “(c) ” Chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be “(c) .”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and cdID. You may think of this value as the offset to the end of the chunk.
notice	Bytes	ASCII characters (\$20–\$7F) representing a copyright notice for the voice or collection of voices. There should be ckSize characters.

The ANNO Chunk

ckID	4 Bytes	The ID for this chunk. These four bytes must be “ANNO.”
ckSize	Rev. Long	The length of this chunk, excluding ckSize and cdID. You may think of this value as the offset to the end of the chunk. Note that this is a Reverse Long; the bytes are stored high byte first.
author	Bytes	ASCII characters (\$20–\$7F) representing the name of the author of the voices or collection of voices. There should be ckSize characters.

Other Chunk Types

There are many types of IFF chunks other than those described in this document. New chunks may be added to ASIF files in the future. If an application encounters a chunk it doesn't recognize when reading an ASIF file, it should ignore it. Note that all chunks should be preserved when copying an IFF file.

Figure 1 illustrates a sample ASIF file as a box diagram.

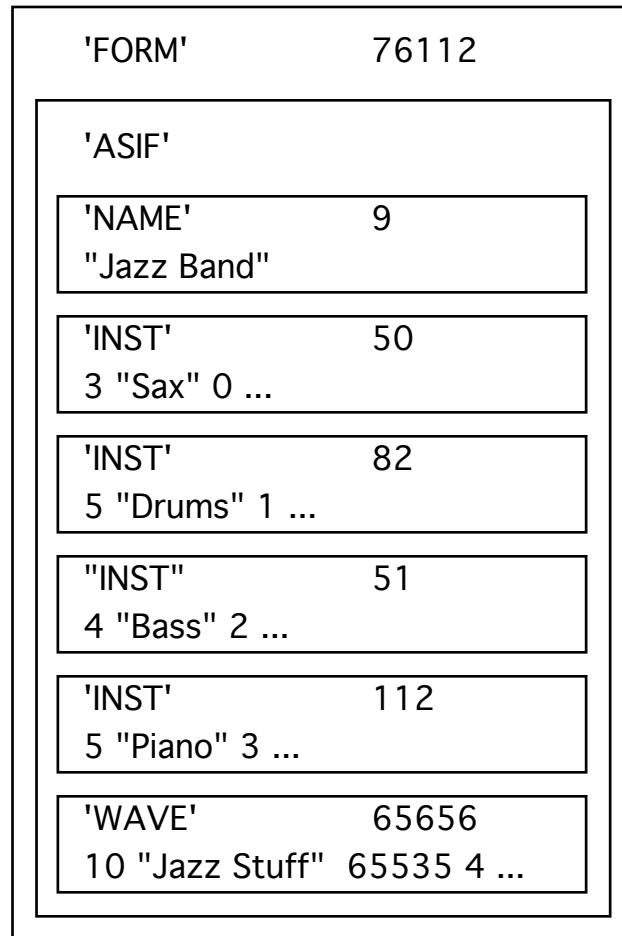


Figure 1—Sample ASIF File

Further Reference

- *Apple IIGS Toolbox Reference Update*
- *Advanced Sampler's Guide* (Ensoniq Corporation)
- "Programming the Ensoniq Mirage," Keyboard Magazine, November 1986
- "EA IFF 85" *Standard for Interchange Format Files*, Electronic Arts, Inc. Describes the underlying conventions for all IFF files.