QuickTime Music Architecture
Atomic Instrument Format In QuickTime 2.1
David Van Brink 19 June 1995
25 July 1995, revised instrument format
7 November 1995, trivia
2 December 1995, instrument component API

0. Introduction

QuickTime 2.0 featured the brand new “QuickTime Music Architecture”, which allowed QuickTime movies and applications to play musical notes, through a General MIDI synthesizer or through the Macintosh’s built-in speaker. Unfortunately, there was no way to include sounds other than the 40 or so sampled instruments licensed from Roland. Although these sounds are of good quality, they are somewhat stifling, perhaps, creatively.

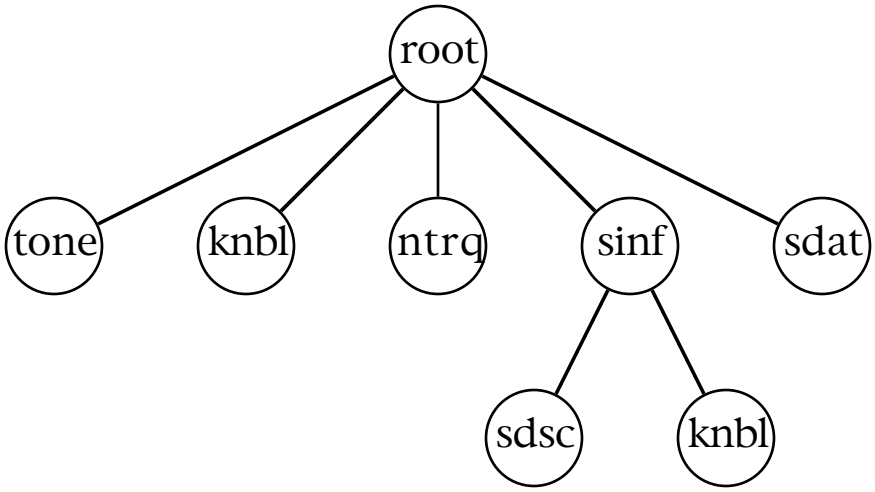
QuickTime 2.2 introduces a public API and format for adding sounds to the QuickTime Music Architecture. These sounds may be embedded in a QuickTime movie, passed via a call to QuickTime, or dropped into the System Folder.

1. Atomic Instrument Format

QuickTime 2.1 includes a set of calls for manipulating *atoms*. These calls allow the construction in memory of hierarchical trees of data, with a fairly simple API set. A tree of atoms lives inside an atom container. There is one and only one root atom per container. Each atom has a four-character (32-bit) type, and a 32-bit ID. Each atom

may be either an internal node, or a leaf atom with data.

This is a picture of an atomic instrument atom container.



The atom types used are as follows.

'tone' - kaiToneDescType
tone description, required
Every atomic instrument must have a tone description.

'knbl' - kaiKnobListType
knob list, optional

```
struct InstKnobRec {
      long     number;
      long     value;
};

struct  InstKnobList{
      long                    knobCount;
      InstKnobRec             knobs[1];
};
```

For a custom instrument, or a modification of the built-in instrument specified by the the tone description, a knob list may specify values for one or more knobs.

'ntrq' - kaiNoteRequestInfoType

note request info, optional

```
struct NoteRequestInfo {
        UInt8 flags;
        UInt8 reserved;                 /* must be zero */
        short polyphony;                /* Maximum number of voices */
        Fixed typicalPolyphony;         /* Hint for level mixing */
};
```

Contains part of a note request structure that is not in the tone description. This is useful when embedding an instrument into the sample description of a QuickTime movie. If it is absent, then "reasonable" values for the polyphony are assumed.

'sinf' - kaiSampleInfoType
sample info, optional: 0, 1, or many

To include your own sampled sounds in an atomic instrument, you must include 1 or more sample info atoms, each of which contains several atoms with various information attached. The sample description atom, in particular, refers via an ID number, to the one or more sample data ('sdat') atoms which must also be present at the level of sibling-to-sample-info.

'sdsc' - kaiSampleDescType
sample description, required, 1 per sample info

```
struct InstSampleDescRec {
        OSType                  dataFormat;
        short                   numChannels;
        short                   sampleSize;
        UnsignedFixed           sampleRate;
        short                   sampleDataID;
        long                    offset;
        long                    numSamples;
        long                    loopType;
        long                    loopStart;
        long    loopEnd;
        long    pitchNormal;
        long    pitchLow;
        long    pitchHigh;
};
```

The sample description atom contains information about how to interpret the sound data, with regards to sample

rate, number of bits per sample, and so on. Note especially the “sampleDataID” field, which determines which of the ‘sdat’ atoms is used for the actual sample data of this instrument.

Presently, the dataFormat field must be ‘raw ‘ for 8 bit audio data (unsigned bytes) and ‘twos’ for 16 bit audio data (signed words).

The three pitch fields specify the range over which to play this particular sample, and the MIDI pitch which is produced when played at the sample rate specified in the sample description atom. Note that by using the knob list atom, to alter the envelope, and the pitch fields, to constrain the sample to a particular keyboard key, one can use the atomic instrument structure to describe complex key splits and drumkits.

‘knbl’ - kaiKnobListType
knob list, optional
Each sample info may optionally override one or more of the instrument knobs. Any knob included in this list overrides the knob settings for the instrument. If the user or score changes a knob (via the MusicSetPartKnob() call) it will have no effect on a sample which overrides that knob.

‘sdat’ - kaiSampleDataType
sample data, optional: 0, 1, or many
Finally, the actual audio data. Rather trivial. It follows the format given by a previous ‘sdsc’ atom. Each sample data atom may be pointed to by more than one sample description atom. A practical use of this might be for a drumkit which has two keys, both of which play an identical closed-high-hat sound.

2. Instrument Component API

When initialized, the software synthesizer searches for components of type ‘inst’. These components may report a list of atomic instruments available to the software synthesizer. At present, ‘inst’ components are only used by the software synthesizer; other synthesizers (music components) do not have this functionality.

There are two calls which an instrument component must implement: GetInstrument(), and GetInfo().

GetInstrument

```
pascal ComponentResult GetInstrument(InstrumentComponent ic,
         long instID, AtomicInstrument *atomicInst)
         ComponentCallNow(1,8);
```

This call returns a handle to an atomic instrument. It is the caller’s responsibility to dispose of it; the instrument component need not keep track of it any further.

GetInfo

```
pascal ComponentResult GetInfo(InstrumentComponent ic,
         InstCompInfoHandle *instInfo)
         ComponentCallNow(2,4);
```

This call returns a handle to a structure which describes all of the instruments that the instrument component may deliver. It is the caller’s responsibility to dispose of it; the instrument component need not keep track of it any further.

The C definition of the structure is as follows.

```
struct GMInstrumentInfo {
    long cmpInstID;
    long gmInstNum;
    long instMatch;
};
```

```
typedef struct GMInstrumentInfo GMInstrumentInfo;
typedef GMInstrumentInfo *GMInstrumentInfoPtr;
typedef GMInstrumentInfo **GMInstrumentInfoHandle;

struct nonGMInstrumentInfoRecord{
    long cmpInstID;           // if 0, category name
    long flags;               // match, show in picker
    long shortNameIndex; // index in shortNames (1 based)
    long longNameIndex;  // index in longNames  (1 based)
};
typedef struct nonGMInstrumentInfoRecord
nonGMInstrumentInfoRecord;

struct nonGMInstrumentInfo {
    long recordCount;
    Handle shortNames;
    Handle longNames;
    nonGMInstrumentInfoRecord instInfo[1];
};
typedef struct nonGMInstrumentInfo nonGMInstrumentInfo;
typedef nonGMInstrumentInfo *nonGMInstrumentInfoPtr;
typedef nonGMInstrumentInfo **nonGMInstrumentInfoHandle;

struct InstCompInfo {
    long infoSize;       // size of this record
    long GMinstrumentCount;
    GMInstrumentInfoHandle GMinstrumentInfo;
    long GMdrumCount;
    GMInstrumentInfoHandle GMdrumInfo;
    long nonGMinstrumentCount;
    nonGMInstrumentInfoHandle nonGMinstrumentInfo;
    long nonGMdrumCount;
    nonGMInstrumentInfoHandle nonGMdrumInfo;
};
typedef struct InstCompInfo InstCompInfo;
typedef InstCompInfo *InstCompInfoPtr;
typedef InstCompInfo **InstCompInfoHandle;
```

3. Software Synthesizer Knobs

Atomic Instruments for the software synthesizer are defined by some waveform data, and a set of “knob” value. These knob values specify things like a volume envelope which is applied to the sampled audio. The knob values appear in atoms with the type

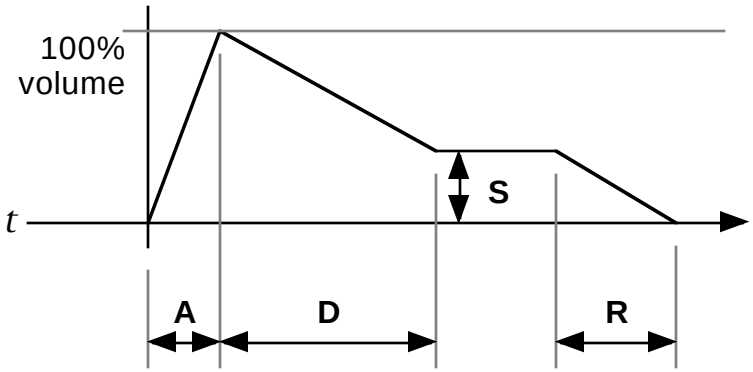
'knbl' as described above. Typically, the instrument will have a full list of knobs, and, if the instrument contains more than a single sample, each sample will contain values for several knobs which are tuned for that particular sample.

Knobs can be specified either by index or by ID. A nonzero value in the high byte of the 24 bit knob number field indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number. ID's are the preferred means to specify a knob, since they will definitely not change over different versions of the QuickTime software; the indices might.

```
kSynthKnobStartID = 0x02000000,
  // Volume related knobs
kSynthKnobVolumeAttackTimeID,
kSynthKnobVolumeDecayTimeID,
kSynthKnobVolumeSustainLevelID,
kSynthKnobVolumeRelease1RateID,
kSynthKnobVolumeRelease1KeyScalingID,
kSynthKnobVolumeRelease2TimeID,

kSynthKnobVolumeLFODelayID,
kSynthKnobVolumeLFORampTimeID,
kSynthKnobVolumeLFOPeriodID,
kSynthKnobVolumeLFOShapeID,
kSynthKnobVolumeLFODepthID,

kSynthKnobVolumeOverallID,
kSynthKnobVolumeVelocity127ID,
kSynthKnobVolumeVelocity96ID,
kSynthKnobVolumeVelocity64ID,
kSynthKnobVolumeVelocity32ID,
kSynthKnobVolumeVelocity16ID,

  // Pitch related knobs
kSynthKnobPitchTransposeID,

kSynthKnobPitchLFODelayID,
kSynthKnobPitchLFORampTimeID,
kSynthKnobPitchLFOPeriodID,
kSynthKnobPitchLFOShapeID,
kSynthKnobPitchLFODepthID,
kSynthKnobPitchLFOQuantizeID,

  // Stereo related knobs
kSynthKnobStereoDefaultPanID,
kSynthKnobStereoPositionKeyScalingID,
kSynthKnobPitchLFOOffsetID,
kSynthKnobExclusionGroupID
```

Volume ADSR Knobs

In traditional synthesizer jargon, “ADSR” stands for Attack-Decay-Sustain-Release, and is a fairly standard notation for describing the change in some attribute of a sound over time. This can be referred to as an “envelope,” and, in the case of volume, the “volume envelope.”



A, **D**, and **R** are specified by a duration of time. **S** is specified as an amplitude level. The **S** (sustain) portion of the envelope lasts as long as the note is held down (in the case of a keyboard, as long as the finger depresses the key).

For a practical example, a piano would have a very short attack time, say, less than 20 milliseconds, and a brief decay time, perhaps 250 milliseconds (a quarter second), decaying to 30 percent of the maximum volume. This nearly instant attack and quick decay would simulate the volume burst at the beginning of an actual piano tone, giving it a percussive sound. The release time determines how quickly the sound disappears once the notes duration is up (or, for a keyboard, how soon the sound disappears after the finger is removed from the key). For a piano, this is pretty quick, perhaps 100 milliseconds.

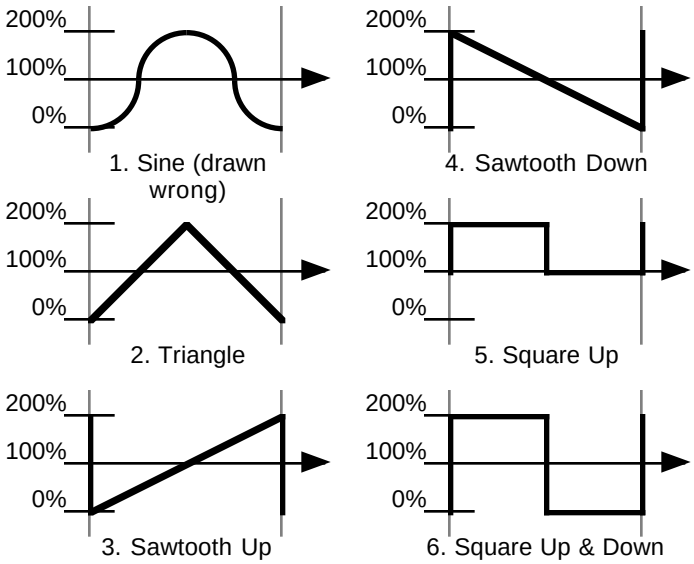
Volume LFO Knobs

“LFO” stands for “Low Frequency Oscillator.” An LFO is used in a music synthesizer to provide a slow and cyclic alteration to some attribute of the sound. In the case of volume, this imparts a pulsating or “tremolo” effect. The user may determine the depth of the effect, with the “Volume LFO Depth” knob, where the minumum setting, 0, causes no LFO modification to the sound, and the maximum setting, 100%, causes the sound to vary between silence

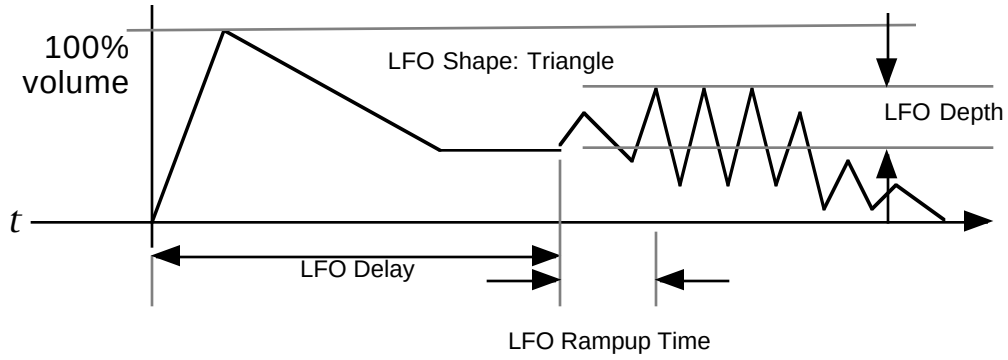
and twice its natural volume.

The rate of the effect is controlled by the “Volume LFO Period” knob, and is specified in milliseconds.

Several shapes of LFO may be applied to the sound. This is controlled by the “Volume LFO Shape” knob.



The LFO effect does not necessarily apply to the entire duration of the sound. It can be controlled further by the “Volume LFO Delay” and “Volume LFO Rampup” times. The delay specified in milliseconds a duration before any LFO is applied at all. The rampup time specifies how long it takes to go from no affect to the full depth specified. Thus, the overall volume envelope, including ADSR and LFO parameters, may resemble the following illustration.



Pitch Transpose and LFO Knobs

The pitch of a sound corresponds to its musical note, or frequency. The Software Synthesizer has several knobs which affect the pitch of each sound played.

Throughout the QuickTime Music Architecture, musical pitches are specified using fixed point MIDI values, where 60.0 represents a piano's middle C key. Eight bits of fraction are used. The standard tuning of a 440 Hz A note is used, where A is MIDI pitch 57.0. Thus, we can convert from MIDI pitch P to frequency in Hz, F, using the equation,

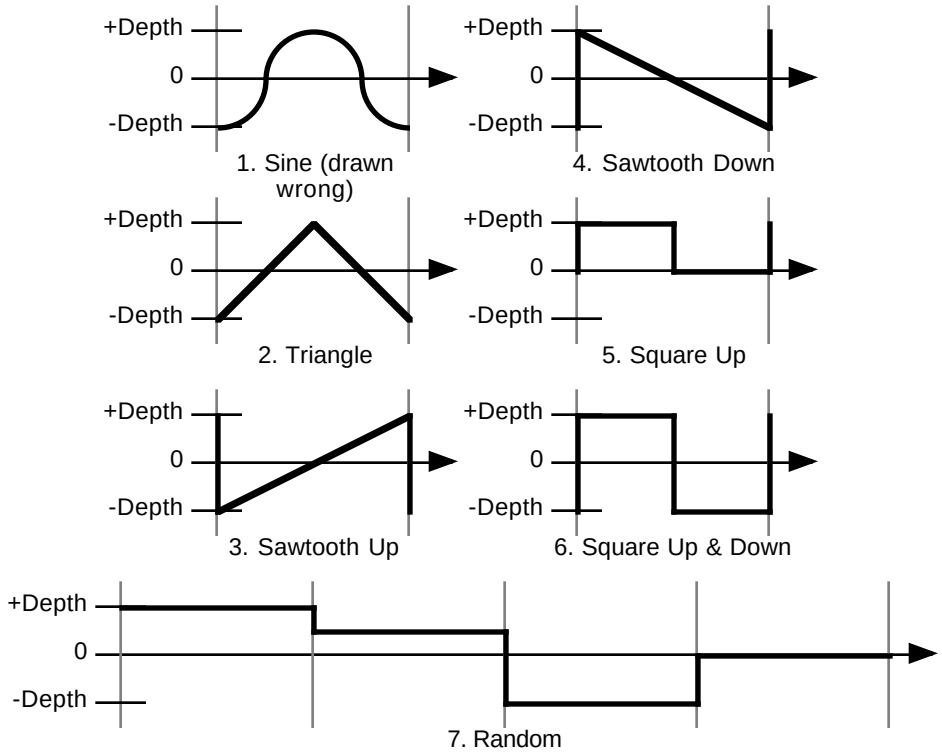
$$F = 440 \times 2^{(P - 57)/12},$$

and from Hz to MIDI pitch using

$$P = 12 \log (F/440)/\log 2 + 57.$$

The "Pitch Transpose" knob simply adds a fixed point value to every note played on the part.

There is also a "Pitch LFO" which modifies the pitch of the sound over time in much the same way that the volume LFO affects the volume. The "Pitch LFO Depth" is specified in fixed point semitones, and the "Pitch LFO Shape" has one additional shape, Random, which picks a random interval up to the current depth setting.




```
  // Pitch related knobs
kSynthKnobPitchTransposeID,

kSynthKnobPitchLFODelayID,
kSynthKnobPitchLFORampTimeID,
kSynthKnobPitchLFOPeriodID,
kSynthKnobPitchLFOShapeID,
kSynthKnobPitchLFODepthID,
kSynthKnobPitchLFOQuantizeID,

  // Stereo related knobs
kSynthKnobStereoDefaultPanID,
kSynthKnobStereoPositionKeyScalingID,
kSynthKnobPitchLFOOffsetID,
kSynthKnobExclusionGroupID
```

VolumeAttackTime
kSynthKnobVolumeDecayTimeID,
kSynthKnobVolumeSustainLevelID,
kSynthKnobVolumeRelease1RateID,
kSynthKnobVolumeRelease1KeyScalingID,
kSynthKnobVolumeRelease2TimeID,

kSynthKnobVolumeLFODelayID,
kSynthKnobVolumeLFORampTimeID,
kSynthKnobVolumeLFOPeriodID,
kSynthKnobVolumeLFOShapeID,
kSynthKnobVolumeLFODepthID,