QuickTime Music Architecture Guide

QuickTime > Audio

2006-01-10

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

Introduction to QuickTime Music Architecture Guide

The QuickTime Music Architecture (QTMA) allows QuickTime movies, applications, and other software to play individual musical notes, sequences of notes, and a broad range of sounds from a variety of instruments and synthesizers. With QTMA, you can also import Standard MIDI files and convert them into a QuickTime movie for easy playback.

Note: This document was previously titled "QuickTime Music Architecture."

You can use the General MIDI component for playing music on a MIDI device attached to a serial port.

Before reading this document, you should already be familiar with QuickTime and QuickTime components.

Important: Developers are encouraged to use the CoreAudio SDK for audio and MIDI application development.

You need to read this document if you are writing an application that creates QuickTime movies and you want to incorporate music tracks as part of the movie, either by importing MIDI files or by programmatically generating musical sequences. If you want to create a music component or add an instrument to the existing library of instruments, you also need to read this document. If you are creating new instruments, you should be familiar with QT atoms and atom containers.

Organization of This Document

This document is presented in two chapters:

- The QuickTime Music Architecture (page 11) describes the features and capabilities of the QuickTime music architecture.
- Using the QuickTime Music Architecture (page 27) describes the functions that allow applications to control all aspects of playing music tracks and generating musical sounds in QuickTime movies.

See Also

The following Apple books cover other aspects of QuickTime programming:

- *QuickTime Overview* gives you the starting information you need to do QuickTime programming.
- *QuickTime Movie Basics* introduces you to some of the basic concepts you need to understand when working with QuickTime movies.

- *QuickTime Movie Creation Guide* describes some of the different ways your application can create a new QuickTime movie.
- *QuickTime Guide for Windows* provides information specific to programming for QuickTime on the Windows platform.
- *QuickTime API Reference* provides encyclopedic details of all the functions, callbacks, data types and structures, atom types, and constants in the QuickTime API.

The QuickTime Music Architecture

This chapter describes the QuickTime music architecture (QTMA), which allows QuickTime movies, applications, and other software to play individual musical notes, sequences of notes, and a broad range of sounds from a variety of instruments and synthesizers. With QTMA, you can also import Standard MIDI files (SMF) and convert them into QuickTime movies for easy playback.

The QuickTime music architecture is implemented as Component Manager components, which is the standard mechanism that QuickTime uses to provide extensibility.

QTMA components exist both in QuickTime for Mac OS X and for Windows.

Different QTMA components are used by a QuickTime movie, depending on if you are playing music or sounds through the computer's built-in audio device, or if you are controlling, for example, a MIDI synthesizer. During playback of a QuickTime movie, the music media handler component isolates your application and the Movie Toolbox from the details of how to actually play a music track. The task of processing the data in a music track is taken care of for you by the media handler through Movie Toolbox calls.

The following sections provide overviews of these components and their capabilities.

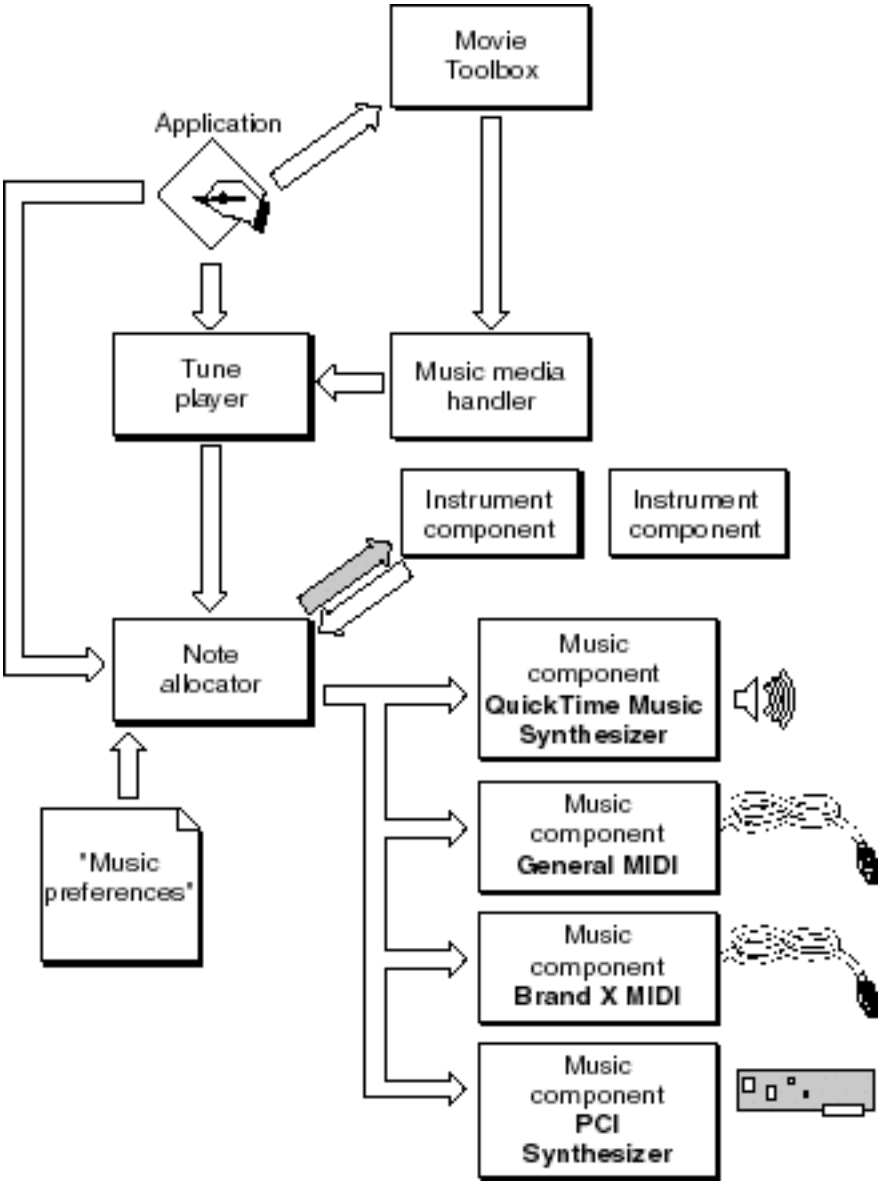
QTMA Components

The QuickTime music architecture includes the following components:

- the note allocator, which plays individual musical notes
- the tune player, which plays sequences of musical notes
- the music media handler, which processes data in music tracks of QuickTime movies
- the General MIDI synthesizer, which plays music on a General MIDI device connected to the computer
- the MIDI synthesizer component, which controls a MIDI synthesizer connected to the computer using a single MIDI channel
- other music components that provide interfaces to specific synthesizers

These components are described in more detail in the following sections. Figure 1-1 illustrates the relationships among the various QTMA components.

Figure 1-1 How QuickTime music architecture components work together



Note Allocator Component

You use the note allocator component to play individual notes. Your application can specify which musical instrument sound to use and exactly which music synthesizer to play the notes on. The note allocator component can also display an Instrument Picker, which allows the user to choose instruments. The note allocator, unlike the tune player, provides no timing-related features to manage a sequence of notes. Its features are similar to a music component, although more generalized. Typically, an application opens a connection to the note allocator, which in turn sends messages to the music component. An application or movie music track can incorporate any number of musical timbres or parts.

To play a single note, your application must open a connection to the note allocator component and call `NANewNoteChannel` with a note request, typically to request a standard instrument within the General MIDI library of instruments. A note channel is similar in some ways to a Sound Manager sound channel in that it needs to be created and disposed of, and can receive various commands. The note allocator provides an application-level interface for requesting note channels with particular attributes. The client specifies the desired polyphony and the desired tone. The note allocator returns a note channel that best satisfies the request.

With an open note channel, an application can call `NAPlayNote` while specifying the note's pitch and velocity. The note is played and continues to play until a second call to `NAPlayNote` is made specifying the same pitch but with a velocity of zero. The velocity of zero causes the note to stop. The note allocator functions let you play individual notes, apply a controller change, apply a knob change, select an instrument based on a required tone, and modify or change the instrument type on an existing note channel.

There are calls for registering and unregistering a music component. As part of registration, the MIDI connections, if applicable, are specified. There is also a call for querying the note allocator for registered music components, so that an application can offer a selection of the existing devices to the user.

Tune Player Component

The tune player component can accept entire sequences of musical notes and play them start to finish, asynchronously, with no further need for application intervention. It can also play portions of a sequence. An additional sequence or sequence section may be queued-up while one is currently being played. Queuing sequences provides a seamless way to transition between sections.

The tune player negotiates with the note allocator to determine which music component to use and allocates the necessary note channels. The tune player handles all aspects of timing, as defined by the sequence of music events. For more information about music events and the event sequence that is required to produce music in a QuickTime movie track, see the section QuickTime Music Events (page 16).

The tune player also provides services to set the volume and to stop and restart an active sequence.

If your application simply wants to play background music, it may be easier to use the QuickTime Movie Toolbox, rather than call the tune player directly.

Music Components Included in QuickTime

Individual music components act as device drivers for each type of synthesizer attached to a particular computer. These music components are included in QuickTime:

- the General MIDI synthesizer component, for playing music on a General MIDI device attached to a serial port.
- the MIDI synthesizer component, which allows QuickTime to control a synthesizer that is connected to a single MIDI channel.

Developers can add other music components for specific hardware and software synthesizers.

Applications do not usually call music components directly. Instead, the note allocator or tune player handles music component interactions. Music components are mainly of interest to application developers who want to access the low-level functionality of synthesizers and for developers of synthesizers (internal cards, MIDI devices, or software algorithms) who want to make the capabilities of their synthesizers available to QuickTime.

In order for an application to call a music component directly, you must first allocate a note channel and then use `NAGetNoteChannelInfo` and `NAGetRegisteredMusicDevice` to get the specific music component and part number.

You can use music component functions to

- obtain specific information about a synthesizer
- find an instrument that best fits a requested type of sound
- play a note with a specified pitch and volume
- change knob values to alter instrument sounds

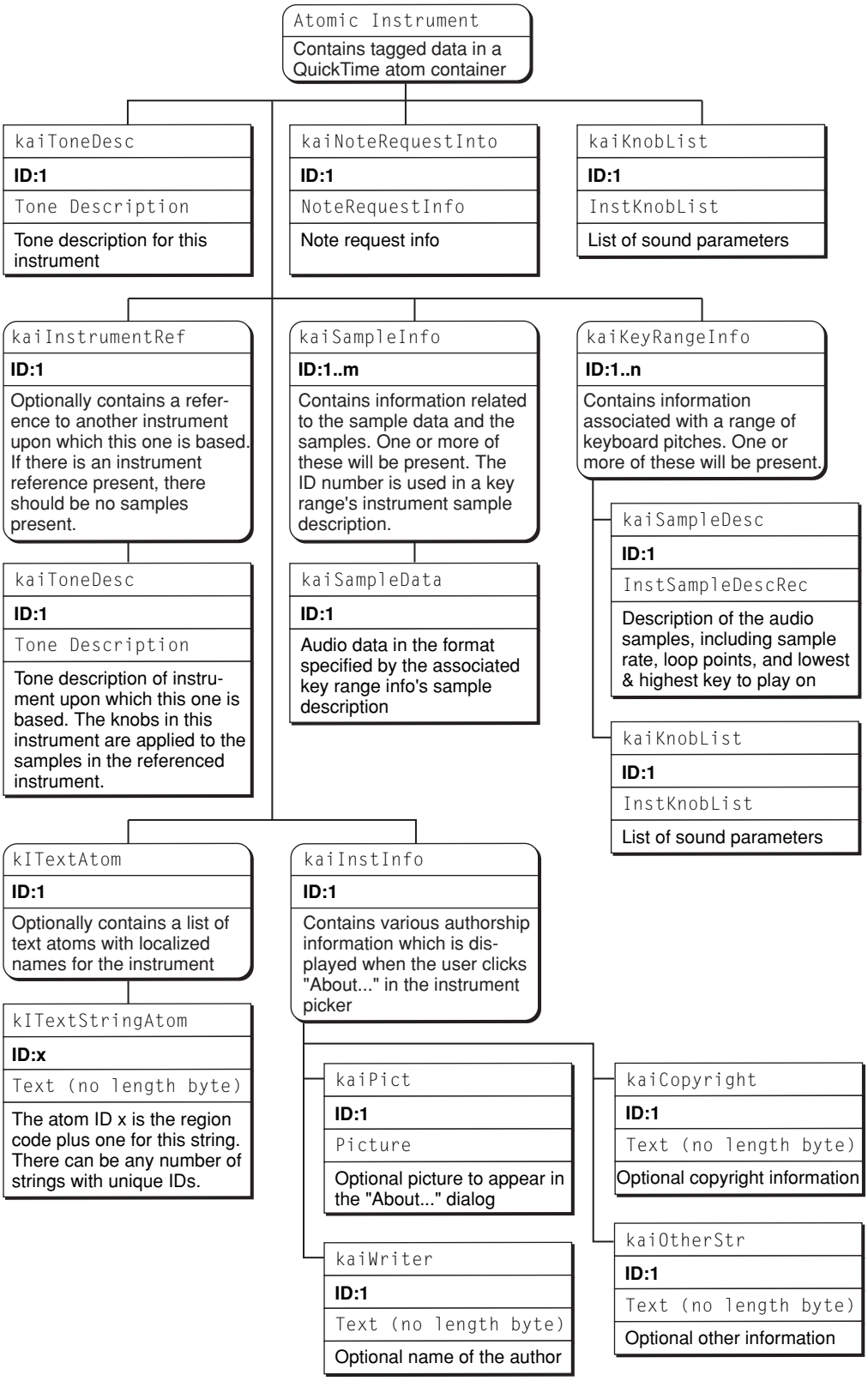
Other functions are for handling instruments and synthesizer parts. You can use these functions to initialize a part to a specified instrument and to get lists of available instrument and drum kit names. You can also get detailed information about each instrument from the synthesizer and get information about and set knobs and controllers.

Instrument Components and Atomic Instruments

When initialized, the note allocator searches for components of type `'inst'`. These components may report a list of atomic instruments. They are called atomic instruments because you create them with QT atoms. These sounds can be embedded in a QuickTime movie, passed via a call to QuickTime, or dropped into the Macintosh System Folder.

QuickTime provides a public format for atomic instruments. Using the QuickTime calls for manipulating atoms, you construct in memory a hierarchical tree of atoms with the data that describes the instrument (see Figure 1-2). The 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.

Figure 1-2 An atomic instrument atom container



The Generic Music Component

To use a new hardware or software synthesizer with the QuickTime music architecture, you need a music component that serves as a device driver for that synthesizer and that can play notes on the synthesizer. You can simplify the creation of a music component by using the services of the generic music component.

To create a music component, you create several resources, for which you get much of the data by calling functions of the generic music component, and implement functions that the generic music component calls when necessary. When a music component is a client of the generic music component, it handles only a few component calls from applications and more relatively simple calls from the generic music component.

MIDI Components

A MIDI component provides a standard interface between the note allocator component and a particular MIDI transport system. The MIDI component supports both input and output of MIDI streams.

Hardware and software developers can provide additional MIDI components. For example, the developer of a multiport serial card can provide a MIDI component that supports direct MIDI input and output using the card. Other MIDI components can support MIDI transport systems for operating systems other than the Mac OS.

QuickTime Music Events

This section describes the data structure of QuickTime music events. The events described here are used to initialize and modify sound-producing music devices and define the notes and rests to be played. Several different event types are defined.

Music events specify the instruments and notes of a musical composition. A group of music events is called a sequence. A sequence of events may define a range of instruments and their characteristics and the notes and rests that, when interpreted, produce the musical composition.

The event sequence required to produce music is usually contained in a QuickTime movie track, which uses a media handler to provide access to the tune player, or an application, which passes them directly to the tune player. QuickTime interprets and plays the music from the sequence data.

The events described in this section initialize and modify sound-producing music devices and define the notes and rests to be played.

Events are constructed as a group of long words. The uppermost 4 bits (nibble) of an event's long word defines its type, as shown in Table 1-1.

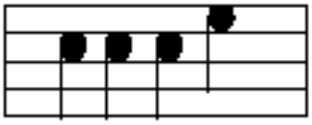
Table 1-1 Event types

First nibble	Number of long words	Event type
000x	1	Rest
001x	1	Note

First nibble	Number of long words	Event type
010x	1	Controller
011x	1	Marker
1000	2	(reserved)
1001	2	Extended note
1010	2	Extended controller
1011	2	Knob
1100	2	(reserved)
1101	2	(reserved)
1110	2	(reserved)
1111	any	General

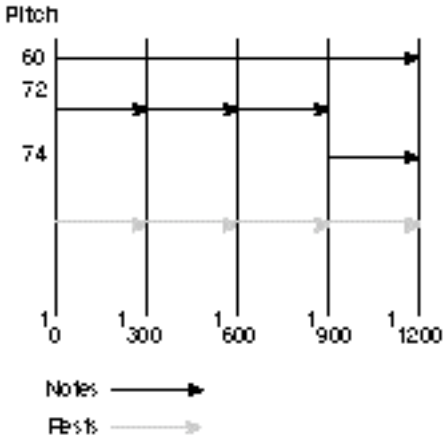
Durations of notes and rests are specified in units of the tune player's time scale (default 1/600 second). For example, consider the musical fragment shown in Figure 1-3.

Figure 1-3 A music fragment



Assuming 120 beats per minute, and a tune player's scale of 600, each quarter note's duration is 300. Figure 1-4 shows a graphical representation of note and rest data.

Figure 1-4 Duration of notes and rests



The general event specifies the types of instruments or sounds used for the subsequent note events. The note event causes a specific instrument, previously defined by a general event, to play a note at a particular pitch and velocity for a specified duration of time.

Additional event types allow sequences to apply controller effects to instruments, define rests, and modify instrument knob values. The entire sequence is closed with a marker event.

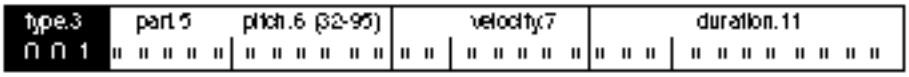
In most cases, the standard note and controller events (two long words) are sufficient for an application's requirements. The extended note event provides wider pitch range and fractional pitch values. The extended controller event expands the number of instruments and controller values over that allowed by a controller event.

The following sections describe the event types in detail.

Note Event

The standard note event (Figure 1-5) supports most music requirements. The note event allows up to 32 parts, numbered 0 to 31, and support pitches from 2 octaves below middle C to 3 octaves above.

Figure 1-5 A note event



Field	Content
note event type	First nibble value = `001X`
Part number	Unique part identifier
Pitch	Numeric value of 0-63, mapped to 32-95
Velocity	0-127, 0 = no audible response (but used to indicate a NOTE OFF)
Duration	Specifies how long to play the note in units defined by the media time scale or tune player time scale

The part number bit field contains the unique part identifier initially used during the `TuneSetHeader` call.

The pitch bit field allows a range of 0-63, which is mapped to the values 32-95 representing the traditional equal tempered scale. For example, the value 28 (mapped to 60) is middle C.

The velocity bit field allows a range of 0-127. A velocity value of 0 produces silence.

The duration bit field defines the number of units of time during which the part will play the note. The units of time are defined by the media time scale or tune player time scale.

Use this macro call to stuff the note event's long word:

```
qtma_StuffNoteEvent(x, instrument, pitch, volume, duration)
```

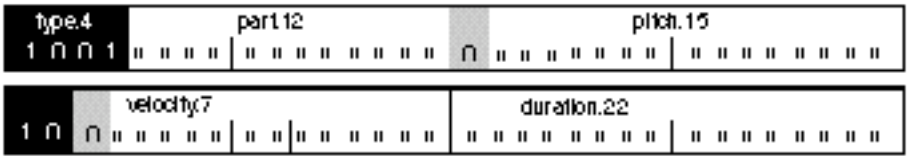
Use these macro calls to extract fields from the note event's long word:

```
qtma_Instrument(x)
qtma_NotePitch(x)
qtma_NoteVelocity(x)
qtma_NoteVolume(x)
qtma_NoteDuration(x)
```

Extended Note Event

The extended note event (Figure 1-6) provides a wider range of pitch values, microtonal values to define any pitch, and extended note duration. The extended note event requires two long words; the standard note event requires only one.

Figure 1-6 An extended note event



Field	Content
Extended note event type	First nibble value = `1001`
Part number	Unique part identifier
Pitch	0-127 standard pitch, 60 = middle C `0x01.00` ... `0x7F.00` allowing 256 microtonal divisions between each notes in the traditional equal tempered scale
Velocity	0-127 where 0 = no audible response (but used to indicate a NOTE OFF)
Duration	Specifies how long to play the note in units defined by media time scale or tune player time scale
Event tail	First nibble of last word = `10XX`

The part number bit field contains the unique part identifier initially used during the `TuneSetHeader` call.

If the pitch bit field is less than 128, it is interpreted as an integer pitch where 60 is middle C. If the pitch is 128 or greater, it is treated as a fixed pitch.

Microtonal pitch values are produced when the 15 bits of the pitch field are split. The upper 7 bits define the standard equal tempered note and the lower 8 bits define 256 microtonal divisions between the standard notes.

Use this macro call to stuff the extended note event's long words:

```
qtma_StuffXNoteEvent(w1, w2, instrument, pitch, volume, duration)
```

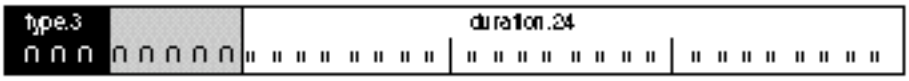
Use these macro calls to extract fields from the extended note event's long words:

```
qtma_XInstrument(m, l)
qtma_XNotePitch(m, l)
qtma_XNoteVelocity(m, l)
qtma_XNoteVolume(m, l)
qtma_XNoteDuration(m, l)
```

Rest Event

The rest event (Figure 1-7) specifies the period of time, defined by either the media time scale or the tune player time scale, until the next event in the sequence is played.

Figure 1-7 A rest event



Field	Content
Rest event type	First nibble value = `000X`
Duration	Specifies the number of units of time until the next note event is played in units defined by media time scale or tune player time scale

Use this macro call to stuff the rest event's long word:

```
qtma_StuffRestEvent(x, duration)
```

Use this macro call to extract the rest event's duration value:

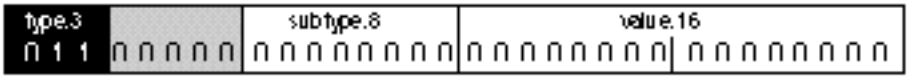
```
qtma_RestDuration(x)
```

Rest events are not used to cause silence in a sequence, but to define the start of subsequent events.

Marker Event

The marker event has three subtypes. The end marker event (Figure 1-8) marks the end of a series of events. The beat marker event marks the beat and the tempo marker event indicates the tempo.

Figure 1-8 A marker event of subtype end



Field	Content
Marker event type	First nibble value = `011X`
Subtype	8-bit unsigned subtype
Value	16-bit signed value

The marker subtype bit field contains zero for an end marker (`kMarkerEventEnd`), 1 for a beat marker (`kMarkerEventBeat`), or 2 for a tempo marker (`kMarkerEventTempo`).

The value bit field varies according to the subtype:

- For an end marker event, a value of 0 means stop; any other value is reserved.
- For a beat marker event, a value of 0 is a single beat (a quarter note); any other value indicates the number of fractions of a beat in 1/65536 beat.
- For a tempo marker event, the value is the same as a beat marker, but indicates that a tempo event should be computed (based on where the next beat or tempo marker is) and emitted upon export.

Use this macro call to stuff the marker event's long word:

```
qtma_StuffMarkerEvent(x, markerType, markerValue)
```

Use these macro calls to extract fields from the marker events long word:

```
qtma_MarkerSubtype(x)
qtma_MarkerValue(x)
```

Controller Event

The controller event (Figure 1-9) changes the value of a controller on a specified part.

Figure 1-9 Controller event

Field	Content
controller event type	First nibble value =`010X`
Part	Unique part identifier
Controller	Controller to be applied to instrument
Value	8.8 bit fixed-point signed controller specific value

For a list of currently supported controller types see Controller Numbers (page 52).

The part field contains the unique part identifier initially used during the `TuneSetHeader` call.

The controller bit field is a value that describes the type of controller used by the part.

The value bit field is specific to the selected controller.

Use this macro call to stuff the controller event's long word:

```
qtma_StuffControlEvent(x, instrument, control, value)
```

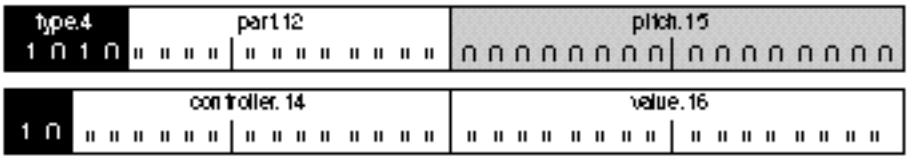
Use these macro calls to extract fields from the controller event's long word:

```
qtma_Instrument(x)
qtma_ControlController(x)
qtma_ControlValue(x)
```

Extended Controller Event

The extended controller event (Figure 1-10) allows parts and controllers beyond the range of the standard controller event.

Figure 1-10 Extended controller event



Field	Content
Extended controller type	First nibble value = `1010`
Part	Instrument index for controller
Controller	Controller for instrument
Value	Signed controller specific value
Event tail	First nibble of last word = `10XX`

The part field contains the unique part identifier initially used during the `TuneSetHeader` call.

The controller bit field contains a value that describes the type of controller to be used by the part.

The value bit field is specific to the selected controller.

Use this macro call to stuff the extended controller event's long words:

```
_StuffXControlEvent(w1, w2, instrument, control, value)
```

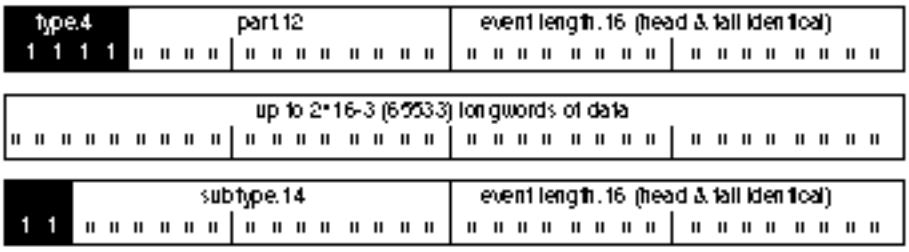
Use these macro calls to extract fields from the extended controller event's long words:

```
qtma_XInstrument(m, l)
qtma_XControlController(m, l)
qtma_XControlValue(m, l)
```

General Event

For events longer than two words, you use the general event with a subtype. Figure 1-11 illustrates the contents of a general event.

Figure 1-11 A note request general event



Field	Content
General event type	First nibble value = `1111`
Part number	Unique part identifier
Event length	Head is number of words in event
Data words	Depends on subtype
Subtype	8-bit unsigned subtype
Event length	tail must be identical to head
Event tail	First nibble of last word = `11XX`

The part number bit field contains a unique identifier that is later used to match note, knob, and controller events to a specific part. For example, to play a note the application uses the part number to specify which instrument will play the note. The general event allows part numbers of up to 12 bits. The standard note and controller events allow part numbers of up to 5 bits; the extended note and extended controller events allow 12-bit part numbers.

The event length bit fields contained in the first and last words of the message are identical and are used as a message format check and to move back and forth through the message. The lengths include the head and tail; the smallest length is 2.

The data words field is a variable length field containing information unique to the subtype of the general event. The subtype bit field indicates the subtype of general event. There are nine subtypes:

- A note request general event (`kGeneralEventNoteRequest`) has a subtype of 1. It encapsulates the note request data structure used to define the instrument or part. It is used in the tune header.

- A part key general event (`KGeneralEventPartKey`) has a subtype of 4. It sets a pitch offset for the entire part so that every subsequent note played on that part will be altered in pitch by the specified amount.
- A tune difference general event (`kGeneralEventTuneDifference`) has a subtype of 5. It contains a standard sequence, with end marker, for the tune difference of a sequence piece. Using a tune difference event is similar to using key frames with compressed video sequences.
- An atomic instrument general event (`kGeneralEventAtomicInstrument`) has a subtype of 6. It encapsulates an atomic instrument. It is used in the tune header. It may be used in place of the `kGeneralEventNoteRequest`.
- A knob general event (`kGeneralEventKnob`) has a subtype of 7. It contains knob ID/knob value pairs. The smallest event is four long words.
- A MIDI channel general event (`kGeneralEventMIDIChannel`) has a subtype of 8. It is used in a tune header. One long word identifies the MIDI channel it originally came from.
- A part change general event (`kGeneralEventPartChange`) has a subtype of 9. It is used in a tune sequence where one long word identifies the tune part that can now take over the part's note channel.
- A no-op general event (`kGeneralEventNoOp`) has a subtype of 10. It does nothing in QuickTime.
- A notes-used general event (`kGeneralEventUsedNotes`) has a subtype of 11. It is four long words specifying which MIDI notes are actually used. It is used in the tune header.

Use these macro calls to stuff the general event's head and tail long words, but not the structures described above:

```
qtma_StuffGeneralEvent(w1, w2, instrument, subType, length)
```

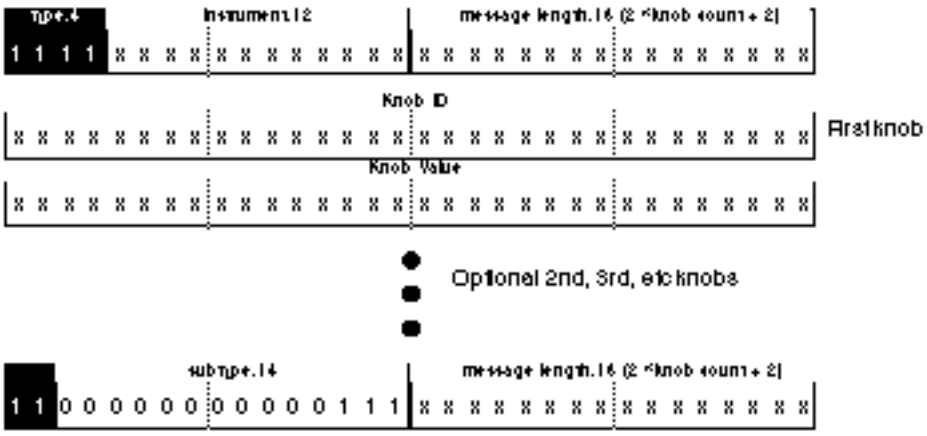
Macros are used to extract field values from the event's head and tail long words.

```
qtma_XInstrument(m, l)
qtma_GeneralSubtype(m, l)
qtma_GeneralLength(m, l)
```

Knob Event

The knob event (Figure 1-12) is used to modify a particular knob or knobs within a specified part.

Figure 1-12 Knob event



Field	Content
Knob event type	First nibble value = `1111` (general event), subtype 7
Length	Length of the event will be 2(#knobs+1)
Part	Unique part identifier
Knob ID	Knob ID within specified part
Knob value	Knob value
Event tail	First nibble of last word = `11XX`, subtype 7

The part field contains the unique part identifier initially used during the `TuneSetHeader` call.

The knob number bit field identifies the knob to be changed.

The 32-bit value composed of the lower 16-bit and upper 16-bit field values is used to alter the specified knob.

The General MIDI Synthesizer Component

The General MIDI synthesizer component controls General MIDI devices. These devices support 24 voices of polyphony, and each of their MIDI channels can access any number of voices.

The MIDI Synthesizer Component

The MIDI synthesizer component allows QuickTime to control a synthesizer connected to a single MIDI channel. It works with any synthesizer that can be controlled through MIDI.

The MIDI synthesizer component does not get information about the synthesizer instruments. Instead, it simply lists available instruments as “Instrument 1,” “Instrument 2,” and so on up to “Instrument 128.”

The Base Instrument Component

When you provide additional sounds for the QuickTime music synthesizer, you can simplify the creation of the necessary instrument resources by using the base instrument component. To create an instrument component, you create a component alias whose target is the base instrument component. The component alias’s data resources specify the capabilities of an instrument, while the code resource of the base instrument component handles all of the component requests sent to the instrument component.

Using the QuickTime Music Architecture

The QuickTime Music Architecture provides functions that allow applications to control all aspects of playing music tracks and generating musical sounds in QuickTime movies.

This chapter discusses a few of the more common operations your application can perform with the QTMA.

Converting MIDI Data to a QuickTime Music Track

QuickTime Pro allows you to open a standard MIDI file and convert it into a QuickTime music track. After the file is converted, the application prompts you to save the converted file as a QuickTime movie. Once saved, a movie controller is displayed and you can play the music.

Importing a Standard MIDI File As a Movie

Most music content exists in Standard MIDI Files (SMF), which have a standard format. All sequencing and composition programs let you save or export files in this format. QuickTime provides facilities for reading an SMF and converting it into a QuickTime movie. During any kind of conversion, the SMF is assumed to be scored for a General MIDI device, and MIDI channel 10 is assumed to be a drum track.

The conversion to a QuickTime movie can happen in one of several ways. Because it is implemented in a QuickTime `'eat '` component, the conversion happens automatically in most cases. Any application that uses the `StandardGetFile` routine to open a movie can also open `'Midi'` files transparently, and can transparently paste Clipboard contents of type `'Midi'` into a movie shown with the standard movie controller.

To explicitly convert a file or handle into a movie, your application can use the Movie Toolbox routines `ConvertFileToMovieFile` and `PasteHandleIntoMovie`, respectively.

When authoring MIDI files to be converted to QuickTime music movies, two MIDI system-exclusive messages can be used for more precise control of the MIDI import process. Note that QuickTime data is divided into media samples. Within video tracks, each video frame is considered one sample; in music tracks, each sample can contain several seconds worth of musical information.

- `F0 11 00 01 xx yy zz F7` sets the maximum size of each media sample to the 21-bit number `xxyyzz`. (MIDI data bytes have the high bit clear, so they have only seven bits of number.) This message can occur anywhere in an SMF.
- `F0 11 00 02 F7` marks an immediate sample break; it ends the current sample and starts a new one. All messages after a sample break message are placed in a new media sample.

Applications can define their own system-exclusive messages of the form `F0 11 7F ww xx yy zz` ... application-defined data ... `F7`, where `ww xx yy zz` is the application's unique signature with the high bits cleared. This is guaranteed not to interfere with Apple's or any other manufacturer's use of system-exclusive codes.

Playing Notes With the Note Allocator

Playing a few notes with the note allocator component is simple and straightforward. To play notes that have a piano sound, for example, you need to open up the note allocator component, allocate a note channel with a request for piano, and play. When you've finished playing notes, you dispose of the note channel and close the note allocator component. The code to accomplish this is shown in Listing 2-2. Before working through the code, you need to look at some important related data structures.

Note-Related Data Structures

A note channel is analogous to a sound channel in that you allocate it, issue commands to it to produce sound, and close it when you're done. To specify details about the note channel, you use a data structure called a `NoteRequest` (see Listing 2-1).

Listing 2-1 Note-related data structures

```
struct NoteRequest {
    NoteRequestInfo  info;
    ToneDescription  tone;
};
struct NoteRequestInfo {
    UInt8    flags;
    UInt8    reserved;
    short    polyphony;
    Fixed    typicalPolyphony;
};
struct ToneDescription {
    OSType     synthesizerType;
    Str31    synthesizerName;
    Str31    instrumentName;
    long     instrumentNumber;
    long     gmNumber;
};
```

The next two fields specify the probable polyphony that the note channel will be used for. Polyphony means, literally, many sounds. A polyphony of 5 means that five notes can be playing simultaneously. The `polyphony` field enables QTMA to make sure that the allocated note channel can play all the notes you need. The typical `polyphony` field is a fixed-point number that should be set to the average number of voices the note channel will play; it may be whole or fractional. Some music components use this field to adjust the mixing level for a good volume. If in doubt, set the typical `polyphony` field to `0X00010000`.

The `ToneDescription` structure is used throughout QTMA to specify a musical instrument sound in a device-independent fashion. This structure's `synthesizerType` and `synthesizerName` fields can request a particular synthesizer to play notes on. Usually, they're set to 0, meaning "choose the best General MIDI synthesizer." The `gmNumber` field indicates the General MIDI (GM) instrument or drum kit sound, which may

be any of 135 such sounds supported by many synthesizer manufacturers. (All these sounds are available on a General MIDI Sound Module.) The GM instruments are numbered 1 through 128, and the seven drum kits are numbered 16385 and higher. For synthesizers that accept sounds outside the GM library, you can use the `instrumentName` and `instrumentNumber` fields to specify some other sound.

Playing Piano Sounds With the Note Allocator

The routine in Listing 2-2 plays notes in a piano sound with the note allocator component.

Listing 2-2 Playing notes with the note allocator component

```
void PlaySomeNotes(void)
{
    NoteAllocator   na;
    NoteChannel     nc;
    NoteRequest     nr;
    ComponentResult  thisError;
    long        t, i;
    na = 0;
    nc = 0;
    // Open up the note allocator.
    na = OpenDefaultComponent(kNoteAllocatorType, 0);
    if (!na)
        goto goHome;
    // Fill out a NoteRequest using NAStuffToneDescription to help, and
    // allocate a NoteChannel.
    nr.info.flags = 0;
    nr.info.reserved = 0;
    nr.info.polyphony = 2;   // simultaneous tones
    nr.info.typicalPolyphony = 0x00010000; // usually just one note
    thisError = NAStuffToneDescription(na, 1, &nr.tone); // 1 is piano
    thisError = NANewNoteChannel(na, &nr, &nc);
    if (thisError || !nc)
        goto goHome;
    // If we've gotten this far, OK to play some musical notes.
    NAPlayNote(na, nc, 60, 80);     // middle C at velocity 80
    Delay(40, &t);          // delay 2/3 of a second
    NAPlayNote(na, nc, 60, 0);   // middle C at velocity 0: end note
    Delay(40, &t);          // delay 2/3 of a second
    // Obligatory do-loop of rising tones
    for (i = 60; i <= 84; i++) {
        NAPlayNote(na, nc, i, 80);    // pitch i at velocity 80
        NAPlayNote(na, nc, i+7, 80);    // pitch i+7 (musical fifth) at
                                        // velocity 80
        Delay(10, &t);                  // delay 1/6 of a second
        NAPlayNote(na, nc, i, 0); // pitch i at velocity 0: end note
        NAPlayNote(na, nc, i+7, 0);  // pitch i+7 at velocity 0:
                                        // end note
    }
goHome:
    if (nc)
        NADisposeNoteChannel(na, nc);
    if (na)
        CloseComponent(na);
```

```
}
```

You start by calling `OpenDefaultComponent` to open a connection to the note allocator. If this routine returns 0, the component wasn't opened, most likely because QTMA wasn't present. Next, you fill in the `NoteRequestInfo` and `ToneDescription` structures, calling the note allocator's `NAStuffToneDescription` routine and passing it the GM instrument number for piano. This routine fills in the `gmNumber` field and also fills in the other `ToneDescription` fields with sensible values, such as the instrument's name in text form in the `instrumentName` field. (The routine can be useful for converting a GM instrument number to its text equivalent.)

After allocating the note channel with `NANewNoteChannel`, you call `NAPlayNote` to play each note. Notice the last two parameters to `NAPlayNote`:

```
ComponentResult NAPlayNote(NoteAllocator na, NoteChannel nc,
    long pitch, long velocity);
```

The value of the `pitch` parameter is an integer from 1 to 127, where 60 is middle C, 61 is C sharp, and 59 is C flat, or B. Similarly, 69 is concert A and is played at a nominal audio frequency of 440 Hz.

The `velocity` parameter's value is also an integer from 1 to 127, or 0. A velocity of 1 corresponds to just barely touching the musical keyboard, and 127 indicates that the key was struck as hard as possible. Different velocities produce tones of different volumes from the synthesizer. A velocity of 0 means the key was released; the note stops or fades out, as appropriate to the kind of sound being played.

You stop the notes at this point after delaying an appropriate amount of time with a call to the `Delay` routine. Finally, you dispose of the note channel and close the note allocator component.

Music Component Functions: Synthesizer

The functions in this section obtain specific information about a synthesizer and obtain a best instrument fit for a requested tone from the available instruments within the synthesizer; play a note with a specified pitch, volume, and duration; get and set a particular synthesizer knob; obtain synthesizer knob information; and get and set external MIDI procedure name entry points.

Knob Flags

Knob flags specify characteristics of a knob. They are used in the `flags` field of a knob description structure. Some flags describe the type of values a knob takes and others describe the user interface. Knob flags are mutually exclusive, so only one should be set (all knob flag constants begin with `kKnobType`).

```
enum {
    kKnobReadOnly               = 16,
    kKnobInterruptUnsafe        = 32,
    kKnobKeyrangeOverride       = 64,
    kKnobGroupStart             = 128,
    kKnobFixedPoint8            = 1024,
    kKnobFixedPoint16           = 2048,
    kKnobTypeNumber             = 0 << 12,
    kKnobTypeGroupName          = 1 << 12,
    kKnobTypeBoolean            = 2 << 12,
    kKnobTypeNote               = 3 << 12,
```

```
    kKnobTypePan                = 4 << 12,
    kKnobTypeInstrument         = 5 << 12,
    kKnobTypeSetting            = 6 << 12,
    kKnobTypeMilliseconds       = 7 << 12,
    kKnobTypePercentage         = 8 << 12,
    kKnobTypeHertz              = 9 << 12,
    kKnobTypeButton             = 10 << 12
};
```

Term	Definition
kKnobReadOnly	The knob value cannot be changed by the user or with a set knob call.
kKnobInterruptUnsafe	Alter this knob only from foreground task time.
kKnobKeyrangeOverride	The knob can be overridden within a single key range (software synthesizer only).
kKnobGroupStart	The knob is first in some logical group of knobs.
kKnobFixedPoint8	Interpret knob numbers as fixed-point 8-bit.
kKnobFixedPoint16	Interpret knob numbers as fixed-point 16-bit.
kKnobTypeNumber	The knob value is a numerical value.
kKnobTypeGroupName	The name of the knob is really a group name for display purposes.
kKnobTypeBoolean	The knob is an on/off knob. If the range of the knob (as specified by the low value and high value in the knob description structure) is greater than one, the knob is a multi-checkbox field.
kKnobTypeNote	The knob value range is equivalent to MIDI keys.
kKnobTypePan	The knob value is the pan setting and is within a range (as specified by the low value and high value in the knob description structure) that goes from left to right.
kKnobTypeInstrument	The knob value is a reference to another instrument number.
kKnobTypeSetting	The knob value is one of n different discrete settings; for example, items on a pop-up menu.
kKnobTypeMilliseconds	The knob value is in milliseconds.
kKnobTypePercentage	The knob value is a percentage of the range.
kKnobTypeHertz	The knob value represents frequency.
kKnobTypeButton	The knob is a momentary trigger push button.

Knob Value Constants

These constants specify unknown or default knob values and are used in various get knob and set knob calls.

```
enum {
    kUnknownKnobValue       = 0x7FFFFFFF,
    kDefaultKnobValue       = 0x7FFFFFFE
};
```

Term	Definition
kUnknownKnobValue	Couldn't find the specified knob value.
kDefaultKnobValue	Set this knob to its default value.

Music Packet Status

These constants are used in the `reserved` field of the MIDI packet structure.

```
enum {
    kMusicPacketPortLost        = 1,
    kMusicPacketPortFound       = 2,
    kMusicPacketTimeGap         = 3
};
```

Term	Definition
kMusicPacketPortLost	The application has lost the default input port.
kMusicPacketPortFound	The application has retrieved the input port from the previous owner.
kMusicPacketTimeGap	The last byte of the packet specifies how long (in milliseconds) to keep the MIDI line silent after sending the packet.

MIDI Packet

The MIDI packet structure describes the data passed by note allocation calls. It is defined by the `MusicMIDIPacket` data type.

```
struct MusicMIDIPacket {
    unsigned short          length;
    unsigned long           reserved;
    UInt8                   data[249];
};
typedef struct MusicMIDIPacket MusicMIDIPacket;
```

Term	Definition
length	The length of the data in the packet.
reserved	This field contains zero or one of the music packet status constants.
data[249]	The MIDI data.

This is the count of data bytes only, unlike MIDI Manager or OMS packets. For information about the music packet status, see Music Packet Status (page 32).

Atomic Instrument Information Flags

These constants specify what pieces of information about an atomic instrument the caller is interested in and are passed to the `MusicGetPartAtomicInstrument` function.

```
enum {
    kGetAtomicInstNoExpandedSamples = 1 << 0,
    kGetAtomicInstNoOriginalSamples = 1 << 1,
    kGetAtomicInstNoSamples         = kGetAtomicInstNoExpandedSamples |
                                      kGetAtomicInstNoOriginalSamples,
    kGetAtomicInstNoKnobList        = 1 << 2,
    kGetAtomicInstNoInstrumentInfo  = 1 << 3,
    kGetAtomicInstOriginalKnobList  = 1 << 4,
    kGetAtomicInstAllKnobs          = 1 << 5
};
```

Term	Definition
`kGetAtomicInstNoExpandedSamples`	Eliminate the expanded samples.
`kGetAtomicInstNoOriginalSamples`	Eliminate the original samples.
`kGetAtomicInstNoSamples`	Eliminate both the original and expanded samples.
`kGetAtomicInstNoKnobList`	Eliminate the knob list.
`kGetAtomicInstNoInstrumentInfo`	Eliminate the About box information.
`kGetAtomicInstOriginalKnobList`	Include the original knob list.
`kGetAtomicInstAllKnobs`	Include the current knob list.

Flags for Setting Atomic Instruments

These flags specify details of initializing a part with an atomic instrument and are passed to the `MusicSetPartAtomicInstrument` function.

```
enum {
    kSetAtomicInstKeepOriginalInstrument    = 1 << 0,
    kSetAtomicInstShareAcrossParts          = 1 << 1,
    kSetAtomicInstCallerTosses              = 1 << 2,
    kSetAtomicInstDontPreprocess            = 1 << 7
};
```

Term	Definition
`kSetAtomicInstKeep-OriginalInstrument`	Keep original sample after expansion.

Term	Definition
kSetAtomicInstShareAcrossParts	Remove the instrument when the application quits.
kSetAtomicInstCallerTosses	The caller isn't keeping a copy of the atomic instrument for later calls to NASetAtomicInstrument.
kSetAtomicInstDontPreprocess	Don't expand the sample. You would only set this bit if you know the instrument is digitally clean or you got it from a MusicGetPartAtomicInstrument call.

Music Component Functions: Instruments and Parts

The functions described in this section initialize a part with an instrument, store instruments, list available instruments, manipulate parts, and get information about parts.

Instrument Info Flags

Use these flags in the MusicGetInstrumentInfo function to indicate which instruments and instrument names you are interested in.

```
enum {
    kGetInstrumentInfoNoBuiltIn         = 1 << 0,
    kGetInstrumentInfoMidiUserInst      = 1 << 1,
    kGetInstrumentInfoNoIText           = 1 << 2
};
```

Term	Definition
kGetInstrumentInfoNoBuiltIn	Don't return built-in instruments.
kGetInstrumentInfoMidiUserInst	Do return user instruments for a MIDI device.
kGetInstrumentInfoNoIText	Don't return international text strings.

Instrument Component Functions

This section describes functions that are implemented by instrument components.

Synthesizer Connection Type Flags

These flags provide information about a MIDI device's connection and are used in the synthesizer connections structure.

```
enum {
    kSynthesizerConnectionMono      = 1,
```

```
    kSynthesizerConnectionMMgr      = 2,
    kSynthesizerConnectionOMS       = 4,
    kSynthesizerConnectionQT        = 8,
    kSynthesizerConnectionFMS       = 16
};
```

Term	Definition
kSynthesizerConnectionMono	If set, and the synthesizer can be both monophonic and polyphonic, the synthesizer is instructed to take up its channels sequentially from the system channel in monophonic mode.
kSynthesizerConnectionMMgr	This connection is imported from the MIDI Manager.
kSynthesizerConnectionOMS	This connection is imported from the Open Music System (OMS).
kSynthesizerConnectionQT	This connection is a QuickTime-only port.
kSynthesizerConnectionFMS	This connection is imported from the FreeMIDI system.

Synthesizer Connections for MIDI Devices

The synthesizer connection structure describes how a MIDI device is connected to the computer. It is defined by the `SynthesizerConnections` data type.

```
struct SynthesizerConnections {
    OSType                      clientID;
    OSType                      inputPortID;
    OSType                      outputPortID;
    long                        midiChannel;
    long                        flags;
    long                        unique;
    long                        reserved1;
    long                        reserved2;
};
typedef struct SynthesizerConnections SynthesizerConnections;
```

Term	Definition
clientID	The client ID provided by the MIDI Manager or `'OMS '` for an OMS port.
inputPortID	The ID provided by the MIDI Manager or OMS for the port used to send to the MIDI synthesizer.
outputPortID	The ID provided by the MIDI Manager or OMS for the port that receives from a keyboard or other control device.
midiChannel	The system MIDI channel or, for a hardware device, the slot number.
flags	Information about the type of connection.
unique	A unique ID you can use instead of an index to identify the synthesizer to the note allocator.

Term	Definition
reserved1	Reserved. Set to 0.
reserved2	Reserved. Set to 0.

For flags values, see Synthesizer Connection Type Flags (page 34).

Instrument Match Flags

These flags are returned in the instMatch field of the General MIDI instrument information structure to specify how QuickTime music architecture matched an instrument request to an instrument.

```
enum {
    kInstrumentExactMatch               = 0x00020000,
    kInstrumentRecommendedSubstitute    = 0x00010000,
    kInstrumentQualityField             = 0xFF000000,
    kRoland8BitQuality          = 0x05000000
};
typedef InstrumentAboutInfo *InstrumentAboutInfoPtr;
typedef InstrumentAboutInfoPtr *InstrumentAboutInfoHandle;
```

Term	Definition
kInstrumentExactMatch	The instrument exactly matches the request.
kInstrumentRecommended-Substitute	The instrument is the approved substitute.
kInstrumentQualityField	The high-order 8 bits of this field specify the quality of the selected instrument. Higher values specify higher quality.
kRoland8BitQuality	For built-in instruments, the value of the high-order 8 bits is always kInstrumentRoland8BitQuality, which corresponds to the quality of an 8-bit Roland instrument.

General MIDI Instrument Information Structure

The General MIDI instrument information structure provides information about a General MIDI instrument within an instrument component. It is defined by the GMInstrumentInfo data type.

```
struct GMInstrumentInfo {
    long                        cmpInstID;
    long                        gmInstNum;
    long                        instMatch;
};
typedef struct GMInstrumentInfo GMInstrumentInfo;
typedef GMInstrumentInfo *GMInstrumentInfoPtr;
typedef GMInstrumentInfoPtr *GMInstrumentInfoHandle;
```

Term	Definition
cmpInstID	The number of the instrument within the instrument component.
gmInstNum	The General MIDI, or standard, instrument number.
instMatch	A flag indicating how the instrument matches the requested instrument.

For instMatch values, see Instrument Match Flags (page 36).

Note Request Constants

These flags specify what to do if the exact instrument requested is not found. They are used in the flags field of the note request information structure.

```
enum {
    kNoteRequestNoGM            = 1,
    kNoteRequestNoSynthType     = 2
};
```

Term	Definition
kNoteRequestNoGM	Don't use a General MIDI synthesizer.
kNoteRequestNoSynthType	Don't use another synthesizer of the same type but with a different name.

Note Request Information Structure

The note request information structure contains information for allocating a note channel that's in addition to that included in a tone description structure. It is defined by the NoteRequestInfo data type.

```
struct NoteRequestInfo {
    UInt8                       flags;
    UInt8                       reserved;
    short                       polyphony;
    Fixed                       typicalPolyphony;
};
typedef struct NoteRequestInfo NoteRequestInfo;
```

Term	Definition
flags	Specifies what to do if the exact instrument requested in a tone description structure is not found.
reserved	Reserved. Set to 0.
polyphony	Maximum number of voices.
typicalPolyphony	Hint for level mixing.

For `flags` values, see Note Request Constants (page 37).

Pick Instrument Flags

The pick instrument flags provide information to the `NAPickInstrument` and `NAPickEditInstrument` functions on which instruments to present for the user to choose from.

```
enum {
    kPickDontMix                = 1,
    kPickSameSynth              = 2,
    kPickUserInsts              = 4,
    kPickEditAllowPick          = 16
};
```

Term	Definition
`kPickDontMix`	Show either all drum kits or all instruments depending on the current instrument. For example, if it's a drum kit, show only drum kits.
`kPickSameSynth`	Show only instruments from the current synthesizer.
`kPickUserInsts`	Show modifiable instruments in addition to ROM instruments.
`kPickEditAllowPick`	Present the instrument picker dialog box. Used only with the `NAPickEditInstrument` function.

Note Allocator Functions: Miscellaneous Interface Tools

The functions in this section provide a user interface for instrument selection and presenting copyright information.

Note Allocator Type

Use these constants to specify the QuickTime note allocator component.

```
enum {
    kNoteAllocatorType              = 'nota'
    kNoteAllocatorComponentType     = 'not2'
};
```

Term	Definition
`kNoteAllocatorType`	The QTMA note allocator type.
`kNoteAllocatorComponentType`	The QTMA note allocator component type.

Tune Queue Depth

This constant represents the maximum number of segments that can be queued with the `TuneQueue` function.

```
enum {
    kTuneQueueDepth     = 8
};
```

Term	Definition
`kTuneQueueDepth`	Deepest you can queue tune segments.

Tune Player Functions

This section describes the functions the tune player provides for setting, queueing, and manipulating music sequences. It also describes tune player utility functions.

Tune Player Type

Use this constant to specify the QuickTime tune player component.

```
enum {
    kTunePlayerType         = 'tune'
};
```

Term	Definition
`kTunePlayerType`	The QuickTime music architecture tune player component type.

Tune Queue Flags

Use these flags in the `TuneQueue` function to give details about how to handle the queued tune.

```
enum {
    kTuneStartNow               = 1,
    kTuneDontClipNotes          = 2,
    kTuneExcludeEdgeNotes       = 4,
    kTuneQuickStart             = 8,
    kTuneLoopUntil              = 16,
    kTuneStartNewMaster         = 16384
};
```

Term	Definition
`kTuneStartNow`	Play even if another tune is playing.
`kTuneDontClipNotes`	Allow notes to finish their durations outside sample.

Term	Definition
kTuneExcludeEdgeNotes	Don't play notes that start at end of tune.
kTuneQuickStart	Leave all the controllers where they are and ignore start time.
kTuneLoopUntil	Loop a queued tune if there is nothing else in the queue.
kTuneStartNewMaster	Start a new master reference timer.

MIDI Component Constants

Use these constants to specify MIDI components.

```
enum {
    kQTMIDIComponentType= FOUR_CHAR_CODE('midi'),
    kOMSComponentSubType= FOUR_CHAR_CODE('OMS '),
    kFMSComponentSubType= FOUR_CHAR_CODE('FMS '),
    kMIDIManagerComponentSubType = FOUR_CHAR_CODE('mmgr')
};
```

Term	Definition
kQTMIDIComponentType	The component type for MIDI components.
kOMSComponentSubType	The component subtype for a Open Music System MIDI component.
kFMSComponentSubType	The component subtype for a FreeMIDI component.
kMIDIManagerComponentSubType	The component subtype for a MIDI Manager component.

MIDI System Exclusive Constants

System exclusive constants can be used to control where sample breaks occur when importing a MIDI file. For more information, see the section Importing a Standard MIDI File As a Movie (page 27).

```
enum {
    kAppleSysexID   = 0x11,
    kAppleSysexCmdSampleSize= 0x0001,
    kAppleSysexCmdSampleBreak= 0x0002,
    kAppleSysexCmdAtomicInstrument = 0x0010,
    kAppleSysexCmdDeveloper= 0x7F00
};
```

MIDI File Import Flags

These flags control the importation of MIDI files.

```
enum {
    kMIDIImportSilenceBefore = 1 << 0,
```

```
    kMIDIImportSilenceAfter = 1 << 1,
    kMIDIImport2OPlayable = 1 << 2,
    kMIDIImportWantLyrics = 1 << 3
};
```

Term	Definition
kMIDIImportSilenceBefore	Specifies to add one second of silence before the first note.
kMIDIImportSilenceAfter	Specifies to add one second of silence after the last note.
kMIDIImport2OPlayable	Specifies to import only MIDI data that can be used with QuickTime. The imported data does not include program changes and has at most 32 parts.
kMIDIImportWantLyrics	Specifies to import karaoke lyrics as a text track.

Part Mixing Flags

Part mixing flags control how a part is mixed with other parts.

```
enum {
    kTuneMixMute= 1,
    kTuneMixSolo= 2
};
```

Term	Definition
kTuneMixMute	Disables the part so that it is not heard.
kTuneMixSolo	Specifies to include only soloed parts in the mix if any parts are soloed.

Atom Types for Atomic Instruments

These constants specify the types of atoms used to build atomic instruments. Atomic instruments are described in Instrument Components and Atomic Instruments (page 14).

```
enum {
    kaiToneDescType                 = 'tone',
    kaiNoteRequestInfoType          = 'ntrq',
    kaiKnobListType                 = 'knbl',
    kaiKeyRangeInfoType             = 'sinf',
    kaiSampleDescType               = 'sdsc',
    kaiSampleDataType               = 'sdat',
    kaiInstRefType                  = 'iref',
    kaiInstInfoType                 = 'iinf',
    kaiPictType                     = 'pict',
    kaiWriterType                   =
```

Instrument Knob Flags

These flags are used in the `knobFlags` field of an instrument knob list structure to indicate what to do if a requested knob is not in the list.

```
enum {
    kInstKnobMissingUnknown         = 0,
    kInstKnobMissingDefault         = 1 << 0
};
```

Term	Definition
`kInstKnobMissingUnknown`	If the requested knob is not in the list, do not set its value.
`kInstKnobMissingDefault`	If the requested knob is not in the list, use its default value.

Loop Type Constants

You can use these constants in the `loopType` field of an atomic instrument sample description structure to indicate the type of loop you want.

```
enum {
    kMusicLoopTypeNormal            = 0,
    kMusicLoopTypePalindrome        = 1
};
```

Term	Definition
`kMusicLoopTypeNormal`	Use a regular loop.
`kMusicLoopTypePalindrome`	Use a back-and-forth loop.

Music Component Type

Use this constant to specify a QuickTime music component.

```
enum {
    kMusicComponentType     = 'musi'
};
```

Term	Definition
`kMusicComponentType`	The type of any QTML music component.

Synthesizer Type Constants

You can use these constants in a tone description structure to specify the type of synthesizer you want to produce the tone.

```
enum {
    kSoftSynthComponentSubType    = 'ss ',
    kGMSynthComponentSubType      = 'gm '
};
```

Term	Definition
kSoftSynthComponentSubType	Use the QuickTime music synthesizer. This is the built-in synthesizer.
kGMSynthComponentSubType	Use the General MIDI synthesizer.

Synthesizer Description Flags

These flags describe various characteristics of a synthesizer. They are used in the `flags` field of the synthesizer description structure.

```
enum {
    kSynthesizerDynamicVoice        = 1,
    kSynthesizerUsesMIDIPort        = 2,
    kSynthesizerMicrotone           = 4,
    kSynthesizerHasSamples          = 8,
    kSynthesizerMixedDrums          = 6,
    kSynthesizerSoftware            = 32,
    kSynthesizerHardware            = 64,
    kSynthesizerDynamicChannel      = 128,
    kSynthesizerHogsSystemChannel   = 256,
    kSynthesizerSlowSetPart         = 1024,
    kSynthesizerOffline             = 4096,
    kSynthesizerGM                  = 16384
};
```

Term	Definition
kSynthesizerDynamicVoice	Voices can be assigned to parts on the fly with this synthesizer (otherwise, polyphony is very important).
kSynthesizerUsesMIDIPort	This synthesizer must be patched through a MIDI system, such as the MIDI Manager or OMS.
kSynthesizerMicrotone	This synthesizer can play microtonal scales.
kSynthesizerHasSamples	This synthesizer has some use for sampled audio data.
kSynthesizerMixedDrums	Any part of this synthesizer can play drum parts.
kSynthesizerSoftware	This synthesizer is implemented in main CPU software and uses CPU cycles.
kSynthesizerHardware	This synthesizer is a hardware device, not a software synthesizer or MIDI device.
kSynthesizerDynamicChannel	This synthesizer can move any part to any channel or disable each part. For devices only.

Term	Definition
kSynthesizerHogsSystemChannel	Even if the kSynthesizerDynamicChannel bit is set, this synthesizer always responds on its system channel. For MIDI devices only.
kSynthesizerSlowSetPart	This synthesizer does not respond rapidly to the various set part and set part instrument calls.
kSynthesizerOffline	This synthesizer can enter an offline synthesis mode.
kSynthesizerGM	This synthesizer is a General MIDI device.

Synthesizer Knob ID Constants

These constants specify knob IDs for the QuickTime music synthesizer. These constants are all of the form kQTMSKnob ID. For example, kQTMSKnobVolumeLFODelayID is the ID constant for the VolumeLFODelay knob.

```
enum {
    kQTMSKnobEnv1AttackTimeID           = 0x02000027,
    kQTMSKnobEnv1DecayTimeID            = 0x02000028,
    kQTMSKnobEnv1ExpOptionsID           = 0x0200002D,
    kQTMSKnobEnv1ReleaseTimeID          = 0x0200002C,
    kQTMSKnobEnv1SustainInfiniteID      = 0x0200002B,
    kQTMSKnobEnv1SustainLevelID         = 0x02000029,
    kQTMSKnobEnv1SustainTimeID          = 0x0200002A,
    kQTMSKnobEnv2AttackTimeID           = 0x0200002E,
    kQTMSKnobEnv2DecayTimeID            = 0x0200002F,
    kQTMSKnobEnv2ExpOptionsID           = 0x02000034,
    kQTMSKnobEnv2ReleaseTimeID          = 0x02000033,
    kQTMSKnobEnv2SustainInfiniteID      = 0x02000032,
    kQTMSKnobEnv2SustainLevelID         = 0x02000030,
    kQTMSKnobEnv2SustainTimeID          = 0x02000031,
    kQTMSKnobExclusionGroupID           = 0x0200001C,
    kQTMSKnobFilterFrequencyEnvelopeDepthID
                                        = 0x0200003B,
    kQTMSKnobFilterFrequencyEnvelopeID  = 0x0200003A,
    kQTMSKnobFilterKeyFollowID          = 0x02000037,
    kQTMSKnobFilterQEnvelopeDepthID     = 0x0200003D,
                                        /* reverb threshhold */
    kQTMSKnobFilterQEnvelopeID          = 0x0200003C,
    kQTMSKnobFilterQID                  = 0x02000039,
    kQTMSKnobFilterTransposeID          = 0x02000038,
    kQTMSKnobLastIDPlus1                = 0x0200003F
    kQTMSKnobPitchEnvelopeDepthID       = 0x02000036, /* filter */
    kQTMSKnobPitchEnvelopeID            = 0x02000035,
    kQTMSKnobPitchLFODelayID            = 0x02000013,
    kQTMSKnobPitchLFODepthFromWheelID   = 0x02000025,
                                        /* volume nnv again */
    kQTMSKnobPitchLFODepthID            = 0x02000017,
    kQTMSKnobPitchLFOOffsetID           = 0x0200001B,
    kQTMSKnobPitchLFOPeriodID           = 0x02000015,
    kQTMSKnobPitchLFOQuantizeID         = 0x02000018,
                                        /* stereo related knobs */
```

```
    kQTMSKnobPitchLFORampTimeID         = 0x02000014,
    kQTMSKnobPitchLFOShapeID            = 0x02000016,
    kQTMSKnobPitchSensitivityID         = 0x02000023,
    kQTMSKnobPitchTransposeID           = 0x02000012,
                                        /* sample can override */
    kQTMSKnobReverbThresholdID          = 0x0200003E,
    kQTMSKnobStartID                    = 0x02000000,
    kQTMSKnobStereoDefaultPanID         = 0x02000019,
    kQTMSKnobStereoPositionKeyScalingID = 0x0200001A,
    kQTMSKnobSustainInfiniteID          = 0x0200001E,
    kQTMSKnobSustainTimeID              = 0x0200001D,
    kQTMSKnobVelocityHighID             = 0x02000021,
    kQTMSKnobVelocityLowID              = 0x02000020,
    kQTMSKnobVelocitySensitivityID      = 0x02000022,
    kQTMSKnobVolumeAttackTimeID         = 0x02000001,
                                        /* sample can override */
    kQTMSKnobVolumeDecayTimeID          = 0x02000002,
                                        /* sample can override */
    kQTMSKnobVolumeExpOptionsID         = 0x02000026, /* env1 */
    kQTMSKnobVolumeLFODelayID           = 0x02000007,
    kQTMSKnobVolumeLFODepthFromWheelID  = 0x02000024,
    kQTMSKnobVolumeLFODepthID           = 0x0200000B,
    kQTMSKnobVolumeLFOPeriodID          = 0x02000009,
    kQTMSKnobVolumeLFORampTimeID        = 0x02000008,
    kQTMSKnobVolumeLFOShapeID           = 0x0200000A,
    kQTMSKnobVolumeLFOStereoID          = 0x0200001F,
    kQTMSKnobVolumeOverallID            = 0x0200000C,
    kQTMSKnobVolumeReleaseKeyScalingID  = 0x02000005,
    kQTMSKnobVolumeReleaseTimeID        = 0x02000006,
                                        /* sample can override */
    kQTMSKnobVolumeSustainLevelID       = 0x02000003,
                                        /* sample can override */
    kQTMSKnobVolumeVelocity127ID        = 0x0200000D,
    kQTMSKnobVolumeVelocity16ID         = 0x02000011,
                                        /* pitch related knobs */
    kQTMSKnobVolumeVelocity32ID         = 0x02000010,
    kQTMSKnobVolumeVelocity64ID         = 0x0200000F,
    kQTMSKnobVolumeVelocity96ID         = 0x0200000E
};
```

Term	Definition
`kQTMSKnobEnv1Attack-TimeID`	Specifies the attack time of the first general-purpose envelope. This is the number of milliseconds between the start of a note and the maximum value of the attack.
`kQTMSKnobEnv1Decay-TimeID`	Specifies the decay time of the first general-purpose envelope. This is the number of milliseconds between the time the attack is completed and the time the envelope level is reduced to the sustain level.

Term	Definition
kQTMSKnobEnv1Exp-OptionsID	Specifies whether segments of the envelope are treated as exponential curves. Bits 0, 1, 2, and 3 of the knob value specify the interpretation of the attack, decay, sustain, and release segments of the envelope, respectively. If any of these bits is 0, the level of the corresponding segment changes linearly from its initial to final value during the time interval specified by the corresponding envelope time knob. If any of these bits is nonzero, the level of the corresponding segment changes exponentially during the time interval specified by the corresponding envelope time knob. During an exponential decrease, the level changes from maximum amplitude (no attenuation) to approximately 1/65536th of maximum amplitude (96 dB of attenuation) during the time interval specified by the corresponding envelope time knob, and afterward the level immediately becomes 0.
kQTMSKnobEnv1Release-TimeID	Specifies the release time of the first general-purpose envelope.
kQTMSKnobEnv1Sustain-InfiniteID	Specifies infinite sustain for the first general-purpose envelope. If the value of this knob is true, the knob overrides the kQTMSKnobEnv1SustainTimeID knob and causes the sustain to last, at undiminished level. Instruments like an organ have infinite sustain.
kQTMSKnobEnv1Sustain-LevelID	Specifies the sustain level of the first general-purpose envelope. This is the percentage of full volume that the sample is initially played at after the decay time has elapsed.
kQTMSKnobEnv1Sustain-TimeID	Specifies the sustain time of the first general-purpose envelope. This is the number of milliseconds it takes for the sample to soften to 90% of its sustain level. This softening occurs in an exponential fashion, so it never actually reaches complete silence. This is used for instruments like a piano, which gradually soften over time even while the key is held down.
kQTMSKnobEnv2Attack-TimeID	Specifies the attack time of the second general-purpose envelope. This is the number of milliseconds between the start of a note and the maximum value of the attack. Percussive sounds usually have zero attack time; gentler sounds may have short attack times. Long attack times are usually used for special effects.
kQTMSKnobEnv2Decay-TimeID	Specifies the decay time of the second general-purpose envelope. This is the number of milliseconds between the time the attack is completed and the time the sample is reduced in volume to the sustain level.

Term	Definition
kQTMSKnobEnv2Exp-OptionsID	Specifies whether segments of the envelope are treated as exponential curves. Bits 0, 1, 2, and 3 of the knob value specify the interpretation of the attack, decay, sustain, and release segments of the envelope, respectively. If any of these bits is 0, the level of the corresponding segment changes linearly from its initial to final value during the time interval specified by the corresponding envelope time knob. If any of these bits is nonzero, the level of the corresponding segment changes exponentially during the time interval specified by the corresponding envelope time knob. During an exponential decrease the level changes from maximum amplitude (no attenuation) to approximately 1/65536th of maximum amplitude (96 dB of attenuation) during the time interval specified by the corresponding envelope time knob, and afterward the level immediately becomes 0.
kQTMSKnobEnv2Release-TimeID	Specifies the release time of the second general-purpose envelope. This is the number of milliseconds it takes for the sound to soften down to silence after the key is released.
kQTMSKnobEnv2Sustain-InfiniteID	Specifies infinite sustain for the second general-purpose envelope. If the value of this knob is true, the knob overrides the kQTMSKnobEnv2SustainTimeID knob and causes the sustain to last, at undiminished volume, until the end of the sample. Instruments like an organ have infinite sustain.
kQTMSKnobEnv2Sustain-LevelID	Specifies the sustain level of the first general-purpose envelope. This is the percentage of full volume that the sample is initially played at after the decay time has elapsed.
kQTMSKnobEnv2Sustain-TimeID	Specifies the sustain time of the second general-purpose envelope. This is the number of milliseconds it takes for the sample to soften to 90% of its sustain level. This softening occurs in an exponential fashion, so it never actually reaches complete silence. This is used for instruments like a piano, which gradually soften over time even while the key is held down.
kQTMSKnobExclusion-GroupID	Specifies an exclusion group. Within an instrument, no two notes with the same exclusion group number, excepting exclusion group, will ever sound simultaneously.This knob is generally used only as an override knob within a key range. (Note that the key range is not an entire instrument.) It is useful for simulating certain mechanical instruments in which the same mechanism produces different sounds. For example, in a drum kit, the open high hat and the closed high hat are played on the same piece of metal. If you assign both sounds to the same exclusion group, playing a closed high hat sound immediately silences any currently playing open high hat sounds.
kQTMSKnobFilter-FrequencyEnvelope-DepthID	Controls the depth of the envelope for the filter frequency. This is an 8.8 signed fixed-point value that specifies the number of semitones the frequency is altered when its envelope (specified by the kQTMSKnobFilter-FrequencyEnvelopeID knob) is at maximum amplitude. If the value of the kQTMSKnobFilterFrequencyEnvelopeID knob is 0, which specifies not to use an envelope to affect filter frequency, the kQTMSKnobFilter-FrequencyEnvelopeDepthID knob is ignored.

Term	Definition
kQTMSKnobFilter-FrequencyEnvelopeID	Specifies which of the two general-purpose envelopes to use to affect the filter frequency, or not to use an envelope to affect filter frequency. If the value of this knob is 0, no envelope is used. If the value of this knob is 1 or 2, the corresponding general-purpose envelope is used.
kQTMSKnobFilter-KeyFollowID	Specifies how closely the frequency of the filter follows the note being played. The emphasis note is determined by the following formula, expressed in MIDI notes: EmphasisNote = (PlayedNote - 60) * (kQTMSKnobFilter-KeyFollowID / 100) - 60 - kQTMSKnobFilterTransposeID.
kQTMSKnobFilter-QEnvelopeDepthID	Controls the depth of the envelope for the emphasis ("Q") of the filter. This is an 8.8 signed fixed-point value that specifies the emphasis is altered when its envelope (specified by the kQTMSKnobFilterQEnvelopeID knob) is at maximum amplitude. If the value of the kQTMSKnobFilterQEnvelopeID knob is 0, which specifies not to use an envelope to affect filter frequency, the kQTMSKnobFilterQEnvelopeDepthID knob is ignored.
kQTMSKnobFilter-QEnvelopeID	Specifies which of the two general-purpose envelopes to use to affect the emphasis ("Q") of the filter, or not to use an envelope to affect the emphasis. If the value of this knob is 0, no envelope is used. If the value of this knob is 1 or 2, the corresponding general-purpose envelope is used.
kQTMSKnobFilterQID	Specifies the emphasis ("Q") of the filter. The value must be in the range 0 to 65536, inclusive, where 0 specifies no emphasis and disables the filter, and 65536 specifies relatively steep emphasis, but not so steep that it approaches feedback.
kQTMSKnobFilter-TransposeID	Specifies a transposition, in semitones, of the frequency of the filter. The emphasis note is determined by the following formula: EmphasisNote = (PlayedNote - 60) * (kQTMSKnobFilterKeyFollowID / 100) - 60 - kQTMSKnobFilterTransposeID.
kQTMSKnobPitch-EnvelopeDepthID	Specifies the depth of the pitch envelope. This is an 8.8 signed fixed-point value that specifies the number of semitones the pitch is altered when the envelope for the pitch (specified by the kQTMSKnobPitchEnvelopeID knob) is at maximum amplitude. If the value of the kQTMSKnobPitchEnvelopeID knob is 0, which specifies not to use an envelope to affect pitch, the kQTMSKnobPitchEnvelopeDepthID knob is ignored.
kQTMSKnobPitch-EnvelopeID	Specifies which of the two general-purpose envelopes to use to affect pitch, or not to use an envelope to affect pitch. If the value of this knob is 0, no envelope is used. If the value of this knob is 1 or 2, the corresponding general-purpose envelope is used to affect pitch.
kQTMSKnobPitch-LFODelayID	Specifies the delay for the pitch LFO. This is the number of milliseconds before the LFO takes effect.

Term	Definition
kQTMSKnobPitch-LFODepthFromWheelID	Specifies the extent to which a synthesizer's modulation wheel (or the MIDI messages it generates) controls the depth of the pitch LFO. The value of this knob is multiplied by the modulation wheel value (a value between 0 to 1), and the result is added to the volume LFO depth specified by the kQTMSKnobPitchLFODepthID knob. Modulation wheel controllers and the MIDI messages they generate are most often used to create vibrato and tremolo effects.
kQTMSKnobPitch-LFODepthID	Specifies the depth of the pitch LFO. This is the number of semitones by which the pitch is altered by the LFO. A value of 0 does not change the pitch. A value of 12 changes the pitch from an octave lower to an octave higher, with one exception: if the square up waveform is used for the LFO, the normal pitch is the minimum pitch.
kQTMSKnobPitch-LFOOffsetID	Specifies the LFO offset. This is a constant value; the units are 8.8 semitones. It is added to the pitch, and is affected by the LFO delay and LFO ramp-up times. It is similar to transposition but subject to the LFO delay and LFO ramp-up times.
kQTMSKnobPitch-LFOPeriodID	Specifies the period for the pitch LFO. This is the wavelength of the LFO in milliseconds. (The LFO rate in Hz is 1000 / kQTMSKnobPitchLFOPeriodID).
kQTMSKnobPitch-LFORampTimeID	Specifies the LFO ramp-up time. This is the number of milliseconds after the LFO delay that it takes for the LFO to reach full effect.
kQTMSKnobPitch-LFOShapeID	Specifies the waveform used for the LFO. The available waveforms are sine, triangle, sawtooth up, sawtooth down, square up, square up-and-down, and random. The sine and triangle shapes both produce a smooth rise and fall of the pitch. The sawtooth up produces a gradual increase in pitch followed by a sudden fall. The sawtooth down shape produces a sudden increase in pitch, followed by a gradual reduction. The square up and square up-and-down shapes apply a sudden pulsing to the pitch; the square up only makes the pitch higher, while the up-and-down variant makes the sound higher and lower. The random shape applies random changes to the pitch, once per LFO period
kQTMSKnobPitch-SensitivityID	Specifies the pitch key scaling. This determines how much the pitch of the struck note affects the pitch of the played note. Typically, this is 100%, meaning that a change in 1 semitone of the struck note produces a change in 1 semitone of the played note. Setting this knob to zero causes every note to play at the same pitch. Setting it to 50% allows for all notes within the quarter-tone scale (24 notes per octave) to be played.
kQTMSKnobPitch-TransposeID	Specifies a transposition for pitches. The value is the number of semitones to transpose; a positive value raises the pitch anda negative value lowers it. The value can be a real number; the fractional part of the value alters the pitch by an additional fraction of a semitone. For example, to raise the pitch of every note played on the instrument by an octave, set the transpose knob to 12.0.
kQTMSKnobStereo-DefaultPanID	Specifies the default pan position for stereo sound. If no pan controller is applied, this determines where in the stereo field notes for this instrument are played

Term	Definition
kQTMSKnobStereo-PositionKeyScalingID	Specifies the key scaling for stereo sound. Amount to modify the stereo placement of notes based upon pitch. At the highest setting, high pitched notes are placed completely in the right speaker, while low pitched notes are placed entirely in the left speaker.
kQTMSKnobSustain-InfiniteID	Specifies infinite sustain for the volume envelope. If the value of this knob is true, the knob overrides the kQTMSKnobSustainTimeID knob and causes the sustain to last, at undiminished volume, until the end of the sample. Instruments like an organ have infinite sustain.
kQTMSKnobSustain-TimeID	Specifies the sustain time of the volume envelope. This is the number of milliseconds it takes for the note to soften to 90% of its sustain level. This softening occurs in an exponential fashion, so it never actually reaches complete silence. This is used for instruments like a piano, which gradually soften over time even while the key is held down.
kQTMSKnobVelocity-HighID	Specifies the maximum velocity value that produces sound for a particular note. If the velocity value is greater, the note does not sound. This can be used to assign different samples to be played for selected velocity ranges.
kQTMSKnobVelocity-LowID	Specifies the minimum velocity value that produces sound for a particular note. If the velocity value is less, the note does not sound. This can be used to assign different samples to be played for selected velocity ranges.
kQTMSKnobVelocity-SensitivityID	Specifies velocity sensitivity, which determines how much the key velocity affects the volume of the note. This value is a percentage. At 100%, a velocity of 1 is nearly silent, and a velocity of 127 is full volume. At 50%, the volume range is from one fourth to three fourths. At 0%, any velocity of key strike produces a half volume note. If the value of this knob is negative, then the note plays more softly as the key is struck harder.
kQTMSKnobVolume-AttackTimeID	Specifies the attack time for the volume envelope. This is the number of milliseconds between the start of a note and maximum volume. Percussive sounds usually have zero attack time; gentler sounds may have short attack times. Long attack times are usually used for special effects.
kQTMSKnobVolume-DecayTimeID	Specifies the decay time for the volume envelope. This is the number of milliseconds between the time the attack is completed and the time the volume is reduced to the sustain level.
kQTMSKnobVolume-ExpOptionsID	Specifies whether segments of the volume envelope are treated as exponential curves. Bits 0, 1, 2, and 3 of the knob value specify the interpretation of the attack, decay, sustain, and release segments of the volume envelope, respectively. If any of these bits is 0, the volume level of the corresponding segment changes linearly from its initial to final value during the time interval specified by the corresponding envelope time knob. If any of these bits is nonzero, the volume level of the corresponding segment changes exponentially during the time interval specified by the corresponding envelope time knob. During an exponential decrease the volume level changes from full volume (no attenuation) to approximately 1/65536th of full volume (96 dB of attenuation) during the time interval specified the corresponding envelope time knob, and afterward the volume level immediately becomes 0.

Term	Definition
kQTMSKnobVolume-LFODelayID	Specifies the delay for the volume LFO. This is the number of milliseconds before the LFO takes effect.
kQTMSKnobVolume-LFODepthFromWheelID	Specifies the extent to which a synthesizer's modulation wheel (or the MIDI messages it generates) controls the depth of the volume LFO. The value of this knob is multiplied by the modulation wheel value (a value between 0 to 1), and the result is added to the volume LFO depth specified by the kQTMSKnobVolumeLFODepthID knob. Modulation wheel controllers and the MIDI messages they generate are most often used to create vibrato and tremolo effects.
kQTMSKnobVolume-LFODepthID	Specifies the depth of the volume LFO. This is the amount, expressed as a percentage, by which the volume is altered by the LFO. A value of 0 does not change the volume. A value of 100 changes the volume from complete silence to twice the volume specified by the envelope, with one exception: if the square up waveform is used for the LFO, the normal envelope volume is the minimum volume.
kQTMSKnobVolume-LFOPeriodID	Specifies the period for the volume LFO. This is the wavelength of the LFO in milliseconds. (The LFO rate in Hz is 1000 / kQTMSKnobPitchLFOPeriodID).
kQTMSKnobVolume-LFORampTimeID	Specifies the ramp-up time for the volume LFO. This is the number of milliseconds after the LFO delay has elapsed that it takes for the LFO to reach full effect.
kQTMSKnobVolume-LFOShapeID	Specifies the waveform used for the LFO. The available waveforms are sine, triangle, sawtooth up, sawtooth down, square up, square up-and-down, and random. The sine and triangle shapes both produce a smooth rise and fall of the volume. The sawtooth up produces a gradual increase in volume followed by a sudden fall. The sawtooth down shape produces a sudden increase in volume, followed by a gradual reduction (often heard as a "ting" sound). The square up and square up-and-down shapes apply a sudden pulsing to the volume; the square up only makes the sound louder, while the up-and-down variant makes the sound louder and softer. The random shape applies random changes to the volume, once per LFO period.
kQTMSKnobVolume-LFOStereoID	If the synthesizer is producing stereo output and the value of this knob is 1, the LFO is applied in phase to one of the stereo channels and 180 degrees out of phase to the other. This often causes a "vibration" effect within the stereo field.
kQTMSKnobVolume-OverallID	Specifies the overall volume of the instrument, in decibels. Increasing the value by 6 doubles the maximum amplitude of the signal, increasing the value by 12 quadruples it, and so on.
kQTMSKnobVolume-ReleaseKeyScalingID	Specifies the release-time key scaling. Modifies the release time based on the key pitch.
kQTMSKnobVolume-ReleaseTimeID	Specifies the release time of the volume envelope. This is the number of milliseconds it takes for the sound to soften down to silence after the key is released.

Term	Definition
`kQTMSKnobVolume-SustainLevelID`	Specifies the sustain level of the volume envelope. This is the percentage of full volume that a note is initially played at after the decay time has elapsed.

Controller Numbers

The controller numbers used by QuickTime are mostly identical to the standard MIDI controller numbers. These are signed 8.8 values. The full range, therefore, is -128.00 to 127+127/128 (or 0x8000 to 0x7FFF).

All controls default to zero except for volume and pan.

Pitch bend is specified in fractional semitones, which eliminates the restrictions of a pitch bend range. You can bend as far as you want, any time you want.

The last 16 controllers (113-128) are global controllers. Global controllers respond when the part number is given as 0, indicating the entire synthesizer.

```
enum {
    kControllerModulationWheel          = 1,
    kControllerBreath                   = 2,
    kControllerFoot                     = 4,
    kControllerPortamentoTime           = 5,
    kControllerVolume                   = 7,
    kControllerBalance                  = 8,
    kControllerPan                      = 10,
    kControllerExpression               = 11,
    kControllerLever1                   = 16,
    kControllerLever2                   = 17,
    kControllerLever3                   = 18,
    kControllerLever4                   = 19,
    kControllerLever5                   = 80,
    kControllerLever6                   = 81,
    kControllerLever7                   = 82,
    kControllerLever8                   = 83,
    kControllerPitchBend                = 32,
    kControllerAfterTouch               = 33,
    kControllerSustain                  = 64,
    kControllerSostenuto                = 66,
    kControllerSoftPedal                = 67,
    kControllerReverb                   = 91,
    kControllerTremolo                  = 92,
    kControllerChorus                   = 93,
    kControllerCeleste                  = 94,
    kControllerPhaser                   = 95,
    kControllerEditPart                 = 113,
    kControllerMasterTune               = 114
};
```

Term	Definition
`kControllerModulationWheel`	This controller controls the modulation wheel. A modulation wheel adds a periodic change to the volume or pitch of a sounding tone, depending on the modulation depth knobs.

Term	Definition
kControllerBreath	This controller controls breath.
kControllerFoot	This controller controls the foot pedal.
kControllerPortamentoTime	This controller adjusts the slur between notes. Set the time to 0 to turn off portamento; there is no separate control to turn portamento on and off.
kControllerVolume	This controller controls volume.
kControllerBalance	This controller controls balance between channels.
kControllerPan	This controller controls balance on the QuickTime music synthesizer and some others. Values are 256-512, corresponding to left to right.
kControllerExpression	This controller provides a second volume control.
kControllerLever1 through kControllerLever8	These are all general-purpose controllers.
kControllerPitchBend	This controller bends the pitch. Pitch bend is specified in positive and negative semitones, with 7 bits per fraction.
kController-AfterTouchkController-AfterTouch	This controller controls channel pressure.
kControllerSustain	This controller controls the sustain effect. The value is a Boolean: positive for on, 0 or negative for off.
kControllerSostenuto	This controller controls sostenuto.
kControllerSoftPedal	This controller controls the soft pedal.
kControllerReverb	This controller controls reverb.
kControllerTremolo	This controller controls tremolo.
kControllerChorus	This controller controls the amount of signal to feed to the chorus special effect unit.
kControllerCeleste	This controller controls the amount of signal to feed to the celeste special effect unit.
kControllerPhaser	This controller controls the amount of signal to feed to the phaser special effect unit.
kControllerEditPart	This controller sets the part number for which editing is occurring. For synthesizers that can edit only one part.
kControllerMasterTune	This controller offsets the entire synthesizer in pitch.

Controller Range

These constants specify the maximum and minimum values for controllers.

```
enum {
    kControllerMaximum      = 0x7FFF,
    kControllerMinimum      = 0x8000
};
```

Term	Definition
`kControllerMaximum`	The maximum value a controller can be set to.
`kControllerMinimum`	The minimum value a controller can be set to.

Drum Kit Numbers

These constants specify the first and last drum kit numbers available to General MIDI drum kits.

```
enum {
    kFirstDrumkit   = 16384,
    kLastDrumkit    = (kFirstDrumkit + 128)
};
```

Term	Definition
`kFirstDrumkit`	The first number in the range of drum kit numbers, which corresponds to "no drum kit." The standard drum kit is `kFirstDrumKit+1=16385`.
`kLastDrumkit`	The last number in the range of drum kit numbers.

Tone Fit Flags

These flags are returned by the `MusicFindTone` function to indicate how well an instrument matches the tone description.

```
enum {
    kInstrumentMatchSynthesizerType     = 1,
    kInstrumentMatchSynthesizerName     = 2,
    kInstrumentMatchName                = 4,
    kInstrumentMatchNumber              = 8,
    kInstrumentMatchGMNumber            = 16
};
```

Term	Definition
`kInstrumentMatchSynthesizerType`	The requested synthesizer type was found.
`kInstrumentMatchSynthesizerName`	The particular instance of the synthesizer requested was found.

Term	Definition
kInstrumentMatchName	The instrument name in the tone description matched an appropriate instrument on the synthesizer.
kInstrumentMatchNumber	The instrument number in the tone description matched an appropriate instrument on the synthesizer.
kInstrumentMatchGMNumber	The General MIDI equivalent was used to find an appropriate instrument on the synthesizer.

Data Structures

This section describes the data structures provided by QuickTime music architecture.

Instrument Knob Structure

An instrument knob structure contains information about an instrument knob. It is defined by the `InstKnobRec` data type.

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

Term	Definition
number	A knob ID or index. A nonzero value in the high byte indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number.
value	The value the knob is set to.

Knob Description Structure

A knob description structure contains sound parameter values for a single knob. It is defined by the `KnobDescription` data type.

```
struct KnobDescription {
    Str63                       name;
    long                        lowValue;
    long                        highValue;
    long                        defaultValue;
    long                        flags;
    long                        knobID;
};
typedef struct KnobDescription KnobDescription;
```

Term	Definition
name	The name of the knob.
lowValue	The lowest number you can set the knob to.
highValue	The highest number you can set the knob to.
defaultValue	A value to use for the default.
flags	Various information about the knob.
knobID	A knob ID or index. A nonzero value in the high byte indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number. Use the knob ID to refer to the knob in preference to the knob index, which may change.

For flags values, see Knob Flags (page 30).

Instrument About Information

The instrument About information structure contains the information that appears in the instrument's About box and is returned by the MusicGetInstrumentAboutInfo function. It is defined by the InstrumentAboutInfo data type.

```
struct InstrumentAboutInfo {
    PicHandle                   p;
    Str255                      author;
    Str255                      copyright;
    Str255                      other;
};
typedef struct InstrumentAboutInfo InstrumentAboutInfo;
```

Term	Definition
p	A handle to a graphic for the About box.
author	The author's name.
copyright	The copyright information.
other	Any other textual information.

Instrument Information Structure

The instrument information structure provides identifiers for instruments and is part of the instrument information list. It is defined by the InstrumentInfoRecord data type.

```
struct InstrumentInfoRecord {
    long                        instrumentNumber;
    long                        flags;
    long                        toneNameIndex;
```

```
    long                        itxtNameAtomID;
};
typedef struct InstrumentInfoRecord InstrumentInfoRecord;
```

Term	Definition
`instrumentNumber`	The instrument number. If the number is 0, the name is an instrument category. See Figure 2-1 for the ranges of instrument numbers. If the value of the instrument number is greater than 65536, its value is transient, and the instrument should be identified by name rather than by number except when the value is immediately passed to the `MusicSetPartInstrumentNumber` function.
`flags`	Unused. Must be 0
`toneNameIndex`	The instrument's position in the `toneNames` index stored in the instrument information list this structure is a part of. The index is a one-based index.
`itxtNameAtomID`	The instrument's position in the `itxtNames` index stored in the instrument information list this structure is a part of.

Instrument Information List

An instrument information list contains the list of instruments available on a synthesizer. It is defined by the `InstrumentInfoList` data type.

```
struct InstrumentInfoList {
    long                        recordCount;
    Handle                      toneNames;
    QTAtomContainer             itxtNames;
    InstrumentInfoRecord        info[1];
};
typedef struct InstrumentInfoList InstrumentInfoList;
typedef InstrumentInfoList *InstrumentInfoListPtr;
typedef InstrumentInfoListPtr *InstrumentInfoListHandle;
```

Term	Definition
`recordCount`	The number of structures in the list.
`toneNames`	A string list of the instrument names as specified in their tone descriptions.
`itxtNames`	A list of international text names, taken from the name atoms.
`info[1]`	An array of instrument information structures.

Non-General MIDI Instrument Information Structure

The non-General MIDI information structure provides information about non-General MIDI instruments within an instrument component. It is defined by the `nonGMInstrumentInfoRecord` data type.

```
struct nonGMInstrumentInfoRecord {
```

```
    long                        cmpInstID;
    long                        flags;
    long                        toneNameIndex;
    long                        itxtNameAtomID;
};
typedef struct nonGMInstrumentInfoRecord nonGMInstrumentInfoRecord;
```

Term	Definition
cmpInstID	The number of the instrument within the instrument component. If the ID is 0, the name is a category name.
flags	Not used.
toneNameIndex	The instrument's position in the toneNames index stored in the instrument information list this structure is a part of. The index is a one-based index.
itxtNameAtomID	The instrument's position in the itxtNames index stored in the instrument information list this structure is a part of.

Non-General MIDI Instrument Information List

A non-General MIDI instrument information list contains the list of non-General MIDI instruments supported by an instrument component. It is defined by the nonGMInstrumentInfo data type.

```
struct nonGMInstrumentInfo {
    long                        recordCount;
    Handle                      toneNames;
    QTAtomContainer             itxtNames;
    nonGMInstrumentInfoRecord   instInfo[1];
};
typedef struct nonGMInstrumentInfo nonGMInstrumentInfo;
typedef nonGMInstrumentInfo *nonGMInstrumentInfoPtr;
typedef nonGMInstrumentInfoPtr *nonGMInstrumentInfoHandle;
```

Term	Definition
recordCount	Number of structures in the list.
toneNames	A short string list of the instrument names as specified in their tone descriptions.
itxtNames	A list of international text names, taken from the name atoms.
instInfo[1]	An array of non-General MIDI instrument information structures.

Complete Instrument Information List

The complete instrument information list contains a list of all atomic instruments supported by an instrument component. It is defined by the InstCompInfo data type.

```
struct InstCompInfo {
    long                        infoSize;
```

```
    long                        GMinstrumentCount;
    GMInstrumentInfoHandle      GMinstrumentInfo;
    long                        GMdrumCount;
    GMInstrumentInfoHandle      GMdrumInfo;
    long                        nonGMinstrumentCount;
    nonGMInstrumentInfoHandle   nonGMinstrumentInfo;
    long                        nonGMdrumCount;
    nonGMInstrumentInfoHandle   nonGMdrumInfo;
};
typedef struct InstCompInfo InstCompInfo;
typedef InstCompInfo *InstCompInfoPtr;
typedef InstCompInfoPtr *InstCompInfoHandle;
```

Term	Definition
infoSize	The size of this structure in bytes.
GMinstrumentCount	The number of General MIDI instruments.
GMinstrumentInfo	A handle to a list of General MIDI instrument information structures.
GMdrumCount	The number of General MIDI drum kits.
GMdrumInfo	A handle to a list of General MIDI instrument information structures.
nonGMinstrumentCount	The number of non-General MIDI instruments.
nonGMinstrumentInfo	A handle to the list of non-General MIDI instruments.
nonGMdrumCount	The number of non-General MIDI drum kits.
nonGMdrumInfo	A handle to the list of non-General MIDI drum kits.

QuickTime MIDI Port

This structure provides information about a MIDI port.

```
struct QTMIDIPort {
    SynthesizerConnections      portConnections;
    Str63                       portName;
};
typedef struct QTMIDIPort QTMIDIPort;
```

Term	Definition
portConnections	A synthesizer connections structure.
portName	The name of the output port.

QuickTime MIDI Port List

This structure contains a list of QuickTime MIDI port structures.

```
struct QTMIDIPortList {
    short               portCount;
    QTMIDIPort          port[1];
};
typedef struct QTMIDIPortList QTMIDIPortList;
```

Term	Definition
`portCount`	The number of MIDI ports in the list.
`port`	An array of QuickTime MIDI port structures.

Note Request Structure

A note request structure combines a tone description structure and a note request information structure to provide all the information available for allocating a note channel. It is defined by the `NoteRequest` data type.

```
struct NoteRequest {
    NoteRequestInfo             info;
    ToneDescription             tone;
};
typedef struct NoteRequest NoteRequest;
```

Term	Definition
`info`	A note request information structure.
`tone`	A tone description structure.

Tune Status

The tune status structure provides information on the currently playing tune.

```
struct TuneStatus {
    unsigned long           tune;
    unsigned long           tunePtr;
    TimeValue               time;
    short                   queueCount;
    short                   queueSpots;
    TimeValue               queueTime;
    long                    reserved[3];
};
typedef struct TuneStatus TuneStatus;
```

Term	Definition
`tune`	The currently playing tune.
`tunePtr`	Current position within the playing tune.

Term	Definition
time	Current tune time.
queueCount	Number of tunes queued up.
queueSpots	Number of tunes that can be added to the queue.
queueTime	Total amount of playing time represented by tunes in the queue. This value can be very inaccurate.
reserved[3]	Reserved. Set to 0.

Instrument Knob List

An instrument knob list contains a list of sound parameters. It is defined by the `InstKnobList` data type.

```
struct InstKnobList {
    long                        knobCount;
    long                        knobFlags;
    InstKnobRec                 knob[1];
};
typedef struct InstKnobList InstKnobList;
```

Term	Definition
knobCount	The number of instrument knob structures in the list.
knobFlags	Instructions on what to do if a requested knob is not in the list.
knob[1]	An array of instrument knob structures.

For `knobFlags` values, see Instrument Knob Flags (page 42).

Atomic Instrument Sample Description Structure

A sample description structure contains a description of an audio sample, including sample rate, loop points, and lowest and highest key to play on. It is defined by the `InstSampleDescRec` data type.

```
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;
};
typedef struct InstSampleDescRec InstSampleDescRec;
```

Term	Definition
dataFormat	The data format type. This is either 'twos' for signed data or 'raw ' for unsigned data.
numChannels	The number of channels of data present in the sample.
sampleSize	The size of the sample: 8-bit or 16-bit.
sampleRate	The rate at which to play the sample in unsigned fixed-point 16.16.
sampleDataID	The ID number of a sample data atom that contains the sample audio data.
offset	Set to 0.
numSamples	The number of data samples in the sound.
loopType	The type of loop.
loopStart	Indicates the beginning of the portion of the sample that is looped if the sound is sustained. The position is given in the number of data samples from the start of the sound.
loopEnd	Indicates the end of the portion of the sample that is looped if the sound is sustained. The position is given in the number of data samples from the start of the sound.
pitchNormal	The number of the MIDI note produced if the sample is played at the rate specified in sampleRate.
pitchLow	The lowest pitch at which to play the sample. Use for instruments, such as pianos, that have different samples to use for different pitch ranges.
pitchHigh	The highest pitch at which to play the sample. Use for instruments, such as pianos, that have different samples to use for different pitch ranges.

For loopType values, see Loop Type Constants (page 42).

Synthesizer Description Structure

A synthesizer description structure contains information about a synthesizer. It is defined by the SynthesizerDescription data type.

```
struct SynthesizerDescription {
    OSType                      synthesizerType;
    Str31                       name;
    unsigned long               flags;
    unsigned long               voiceCount;
    unsigned long               partCount;
    unsigned long               instrumentCount;
    unsigned long               modifiableInstrumentCount;
```

```
    unsigned long               channelMask;
    unsigned long               drumPartCount;
    unsigned long               drumCount;
    unsigned long               modifiableDrumCount;
    unsigned long               drumChannelMask;
    unsigned long               outputCount;
    unsigned long               latency;
    unsigned long               controllers[4];
    unsigned long               gmInstruments[4];
    unsigned long               gmDrums[4];
};
typedef struct SynthesizerDescription SynthesizerDescription;
```

Term	Definition
`synthesizerType`	The synthesizer type. This is the same as the music component subtype.
`name`	Text name of the synthesizer type.
`flags`	Various information about how the synthesizer works.
`voiceCount`	Maximum polyphony.
`partCount`	Maximum multi-timbrality (and MIDI channels).
`instrumentCount`	The number of built-in ROM instruments. This does not include General MIDI instruments.
`modifiableInstrument-Count`	The number of slots available for saving user-modified instruments.
`channelMask`	Which channels a MIDI device always uses for instruments. Set to `FFFF` for all channels.
`drumPartCount`	The maximum multi-timbrality of drum parts. For synthesizers where drum kits are separated from instruments.
`drumCount`	The number of built-in ROM drum kits. This does not include General MIDI drum kits. For synthesizers where drum kits are separated from instruments
`modifiableDrumCount`	The number of slots available for saving user-modified drum kits. For MIDI synthesizers where drum kits are separated from instruments
`drumChannelMask`	Which channels a MIDI device always uses for drum kits. Set to `FFFF` for all channels
`outputCount`	The number of audio outputs. This is usually two.
`latency`	Response time in microseconds.
`controllers[4]`	An array of 128 bits identifying the available controllers. Bits are numbered from 1 to 128, starting with the most significant bit of the long word, and continuing to the least significant of the last bit.
`gmInstruments[4]`	An array of 128 bits giving the available General MIDI instruments.

Term	Definition
gmDrums[4]	An array of 128 bits giving the available General MIDI drum kits.

For flags values, see Synthesizer Description Flags (page 43). For controller numbers, see Controller Numbers (page 52).

Tone Description Structure

A tone description structure provides the information needed to produce a specific musical sound. The tune header has a tone description for each instrument used. Tone descriptions are also used in the tone description atoms of atomic instruments. The tone description structure is defined by the ToneDescription data type.

```
struct ToneDescription {
    BigEndianOSType             synthesizerType;
    Str31                       synthesizerName;
    Str31                       instrumentName;
    BigEndianLong               instrumentNumber;
    BigEndianLong               gmNumber;
};
typedef struct ToneDescription ToneDescription;
```

Term	Definition
synthesizerType	The synthesizer type. A value of 0 specifies that any type of synthesizer is acceptable.
synthesizerName	The name of the synthesizer component instance. A value of 0 specifies that the name can be ignored.
instrumentName	The name of the instrument to use.
instrumentNumber	The instrument number of the instrument to use. This value, which must be in the range 1-262143, can specify General MIDI and GS instruments as well as other instruments (see Figure 2-1). The instrument specified by this field is used if it is available; if not, the instrument specified by the gmNumber field is used. If neither of the instruments specified by the instrumentNumber or gmNumber fields is available, the instrument specified by the instrumentName field is used. Finally, if none of these fields specifies an instrument that is available, no tone is played.
gmNumber	The instrument number of a General MIDI or GS instrument to use if the instrument specified by the instrumentNumber field is not available. This value, which must be in the range 1-16383, can specify only General MIDI and GS instruments (see Table 1-11). The instrument specified by the instrumentNumber field is used if it is available; if not, the instrument specified by the gmNumber field is used. If neither of the instruments specified by the instrumentNumber or gmNumber fields is available, the instrument specified by the instrumentName field is used. Finally, if none of these fields specifies an instrument that is available, no tone is played.

For synthesizerType values, see Synthesizer Type Constants (page 42).

GS instruments conform to extensions defined by Roland Corporation to the General MIDI specifications.

Figure 2-1 Instrument number ranges

Name	Low	High	Low (Hex)	High (Hex)
GM Instrument	1	128	0x00000001	0x00000080
GM Drumkit	16385	16512	0x00004001	0x00004080
GS Instrument	128	16383	0x00000081	0x00003FFF
ROM Instrument	32768	65535	0x00008000	0x0000FFFF
User Instrument	65536	131071	0x00010000	0x0001FFFF
Internal Index	131072	262143	0x00020000	0x0003FFFF
All Other Numbers Illegal And Reserved				

Instrument	Definition
GM instrument	An instrument number in this range specifies a standard General MIDI instrument that should sound the same on all synthesizers that support General MIDI.
GM drum kit	An instrument number in this range specifies a standard General MIDI drum kit instrument that should sound the same on all synthesizers that support General MIDI.
GS instrument	An instrument number in this range specifies a standard GS instrument that should sound the same on all synthesizers that support the Roland GS extensions to General MIDI.
ROM instrument	An instrument number in this range specifies an instrument of a synthesizer that not a standard General MIDI or GS instrument.
User instrument	Instruments number in this range are transient and are assigned when necessary for additional instruments, such as instruments in a newly installed GS library or custom instruments for a game. Applications should refer to these additional instruments by name rather by number.
Internal index	An instrument index value returned by the `MusicFindTone` function that can be passed immediately in a call to `MusicSetPartInstrumentNumber`. Values in this range are not persistent and should never be stored or used in any other way.

Result Codes

This section lists all the result codes returned by QuickTime music architecture functions.

Constant	Value	Description
`notImplementedMusicOSErr`	-2071	Call to a routine that is not supported by a particular music component.
`cantSendToSynthesizerOSErr`	-2072	Attempt to use a synthesizer before it has been initialized, given a MIDI port to use, or told which slot card to use. For example, the `MusicSetMIDIProc` function has not been called.

Constant	Value	Description
illegalVoiceAllocationOSErr	-2074	Attempt to allocate more voices than a synthesizer supports.
illegalPartOSErr	-2075	Usually indicates use of a part number parameter outside the range 1...partcount.
illegalChannelOSErr	-2076	Attempt to use a MIDI channel outside the range 1...16.
illegalKnobOSErr	-2077	Attempt to use a knob index or knob ID that is not valid.
illegalKnobValueOSErr	-2078	Attempt to set a knob outside its allowable range, as specified in its knob description structure.
illegalInstrumentOSErr	-2079	Attempt to use an instrument or sound that is not available or there is some other problem with the instrument, such as a bad instrument number.
illegalControllerOSErr	-2080	Attempt to get or set a controller that is outside the allowable controller number range or is not recognized by this particular music component.
midiManagerAbsentOSErr	-2081	Attempt to use MIDI Manager for a synthesizer when the MIDI Manager is not installed.
synthesizerNot-RespondingOSErr	-2082	Various hardware problems with a synthesizer.
synthesizerOSErr	-2083	Software problem with a synthesizer.
illegalNoteChannelOSErr	-2084	Attempt to use a note channel that is not initialized or is otherwise errant.
noteChannelNot-AllocatedOSErr	-2085	It was not possible to allocate a note channel.
tunePlayerFullOSErr	-2086	Attempt to queue up more tune segments (with TuneQueue) than allowed.
tuneParseOSErr	-2087	TuneSetHeader or TuneQueue encountered illegal tune sequence data.

Document Revision History

This table describes the changes to *QuickTime Music Architecture Guide*.

Date	Notes
2006-01-10	Removed obsolete material and changed title from "QuickTime Music Architecture."
2002-09-17	New document that explains how to enable synthesizing and playing of sounds and musical sequences.

Document Revision History