QuickTime Overview

2005-08-11

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

Simultaneously published in the United States and Canada.

Contents

Figures

Chapter 1 **QuickTime Overview 9**

Introduction To QuickTime Overview

This document is intended as a general introduction to QuickTime for programmers at all levels. It is also suitable for non-programmers with a strong technical interest. It is a good starting place for readers with no prior knowledge of QuickTime.

Organization of This Document

This document contains the following sections:

- "QuickTime Overview" (page 9) begins with a nutshell description of QuickTime and its major uses.
- "Architecture" (page 9) describes the tool sets, components, and output of QuickTime.
- "The QuickTime API" (page 15) describes the organization of the API and how to find your way around in it.
- "QuickTime Movies" (page 17) describes the movie data structure and defines important terms used to describe the parts of a movie.
- "Atoms, QuickTime Atoms, and Atom Containers" (page 24) explains how movies are stored in files and how complex QuickTime data structures are created and passed.
- "Streaming, Broadcasting, and Progressive Download" (page 28) describes alternate methods of delivering movies over a network or the Internet.
- "QuickTime Road Map" (page 30) shows how the QuickTime documentation and API are organized and provides links to further reading after you have read the overview.

QuickTime Overview

QuickTime is a cross-platform multimedia architecture for the Mac OS and Windows. It consists of a set of multimedia operating-system extensions (implemented as DLLs in Windows), a comprehensive API, a file format, and a set of user applications such as QuickTime Player, the QuickTime ActiveX control, and the QuickTime browser plug-in.

QuickTime is a complete multimedia architecture, not just a media player. It supports creating, producing, and delivering a broad variety of media. QuickTime provides end-to-end support for the entire process: capturing media in real time; synthesizing media programatically; importing and exporting existing media; editing and compositing; compression, delivery, and user playback.

Specific tasks that QuickTime is useful for include:

- Playing movies and other media, such as Flash or MP3 audio
- Nondestructive editing of movies and other media
- Importing and exporting images between formats, such as JPEG and PNG
- Compressing and decompressing sound and video
- Compositing, layering, and arranging multiple media elements from different sources
- Synchronizing multiple time-dependent media to a single timeline
- Capturing and storing sequences from real-time sources, such as audio and video inputs
- Creating movies programmatically from synthesized data
- Creating sprites that use intelligent, scripted animation
- Creating presentations that interact with viewers, remote databases, and application servers
- Creating movies that include customized window shapes, “skins,” and controls
- Streaming movies in real time over a network or the Internet
- Broadcasting real-time streams from live sources such as cameras and microphones
- Distributing downloadable media on disc or over a network or the Internet

Architecture

The QuickTime programming architecture is a combination of flexible tool sets and plug-in components.

Tool Sets

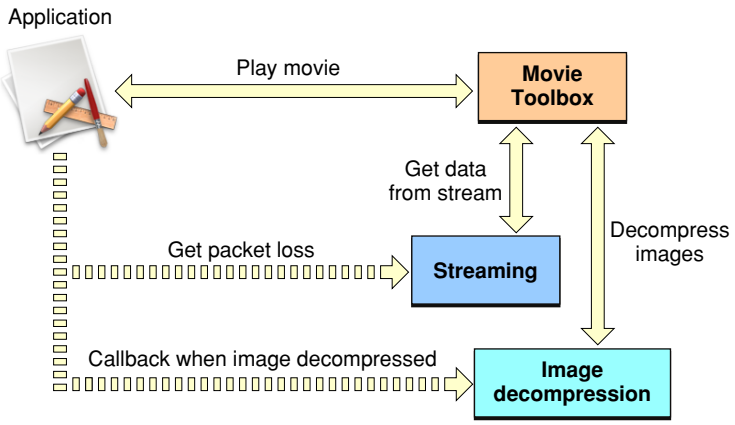
To support the complete spectrum of multimedia tasks, the QuickTime API contains a collection of **toolsets**, such as the **Movie Toolbox,** the **Image Compression Manager,** the **sequence grabber,** and the **QuickTime streaming API.**

- The Movie Toolbox is used to initialize QuickTime; open, play, edit, and save movies; and manipulate time-based media.
- The Image Compression Manager is a device-independent and driver-independent means of compressing and decompressing image data.
- The sequence grabber is a framework for components that capture and record samples from real-time sources, such as video cards or audio inputs.
- The streaming API allows you to send and receive real-time streams using standard protocols such as RTP and RTSP.

There are several other tool sets, including **QuickTime VR,** the **sprite toolbox,** and the **wired movies API,** but you don't need to use them all or even know them all. These tool sets work together, allowing you to focus on the task at hand, without needing to learn the entire QuickTime API. The different tool sets often share data types and programming paradigms, making it relatively easy to extend your knowledge of QuickTime as you go.

Many tool sets are useful when you need direct access to things that QuickTime usually deals with automatically. For example, when you use the Movie Toolbox to play a movie, it may open a stream of real-time data and decompress a series of images, without requiring you to interact with the streaming API or the Image Compression Manager. But if you need to check for streaming packet loss or be notified each time an image is decompressed, you can use the appropriate tool set from your application.

Figure 1-1 Using tool sets



Components

The QuickTime architecture makes extensive use of **components,** making it modular, flexible, and extensible. A QuickTime component is a shared code resource with a defined API. It is possible to add a new component to QuickTime and have existing applications automatically find it and use it when needed, largely because it responds to the same API as existing components of that general type.

For example, QuickTime works with a number of media types: sound, video, text, sprites, Flash, 3D models, photographic virtual reality, and others. Each media type is supported by a media handler component. The number and types of supported media are continually growing. You can add a new media type to QuickTime yourself by creating a new media handler component.

There are also component types for controlling and playing movies, importing and exporting media, compressing and decompressing images or sound, accessing data (from file systems, network servers, or blocks of memory), capturing sequences of digitized sample data, and so on. Here is a partial list:

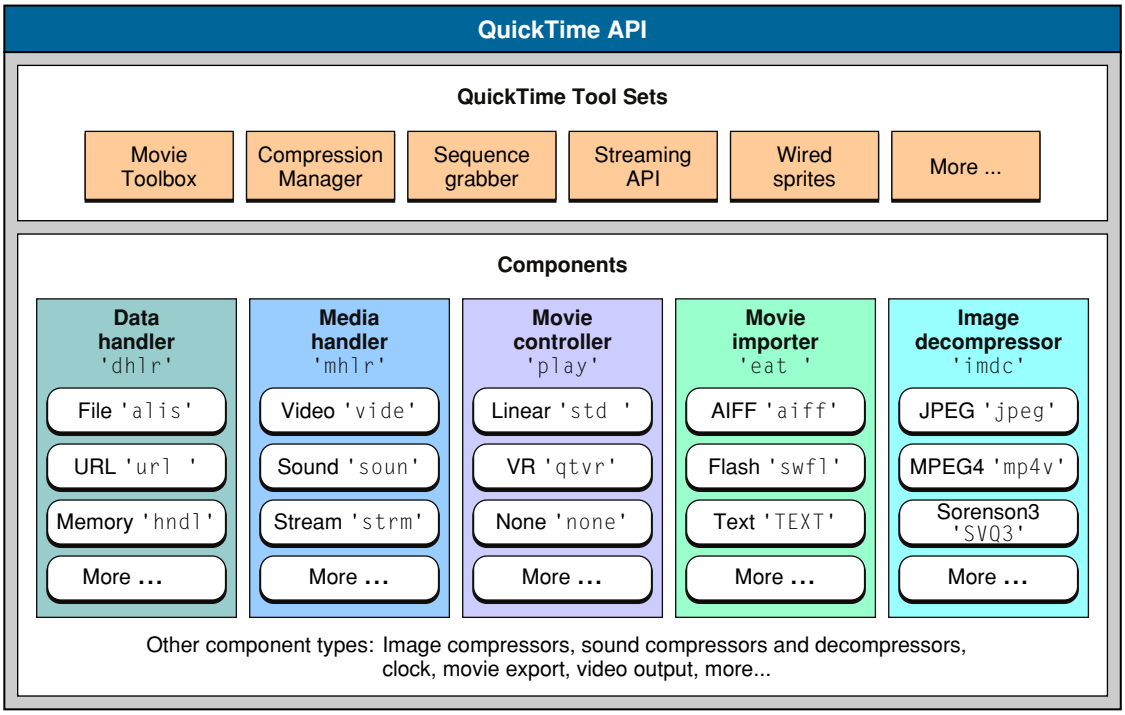
- Movie controller components are used to play movies and can provide a standard user interface.
- Media handler components handle a particular type of media data, such as video, sound, Flash, or text.
- Data handler components access data from a particular kind of data source, such as local files, URLs, or handles.
- Image compressor components compress or decompress image data.
- Image compression dialog components let the user specify the parameters for compression operations.
- Video digitizer components are used to control video digitization by external devices such as video capture cards.
- Movie data-exchange components (also known as movie import and movie export components) move data between QuickTime and other formats. QuickTime can play any type of media file for which it has an importer or create any type for which it has an exporter.
- Video output components convert the visual output of QuickTime movies into video streams for devices other than displays.
- Graphics import and export components provide a single API that lets you work with a wide variety of image file formats.
- Music components process and synthesize MIDI-like music tracks in QuickTime movies.
- Effects and transitions components implement visual filters, effects, and transitions. (Effects components are implemented as a special type of image compressor component.)

As with tool sets, you don't need to work with, or even know about, every type of component. Most components are used automatically as needed, but most also support an API that you can work with directly when you want to.

Each component has a **type,** a **subtype,** and a **manufacturer** code, each represented by a four-character code. A four-character code is a 32-bit value that is usually best interpreted as four ASCII characters. For example, an image decompressor component has a type of `'imdc'`.

QuickTime often has multiple components of a given type. For example, QuickTime has many decompressor components. They all have the same type: `'imdc'`. Each `'imdc'` component has a subtype that specifies the kind of compression it understands, such as JPEG, and a manufacturer code that distinguishes among components of the same subtype. For example, the image decompressor for JPEG supplied by Apple has the type, subtype, and manufacturer codes of `'imdc'`, `'jpeg'`, `'aapl'`.

Figure 1-2 Some commonly used tool sets and components



Important: There are no "three-character" codes. Some of the four-character codes, such as `'eat '`, include an ASCII blank space (`0x020`). The space character is an important part of the code, and cannot be omitted.

QuickTime ships with a number of components and has the ability to download others when needed (provided there is an Internet connection and user consent). Third-party components installed locally can, in many cases, be recognized and used by existing applications without modification.

QuickTime finds, selects, loads, and unloads components as needed. This is often transparent to the applications programmer. For example, when you tell QuickTime to open a movie, QuickTime automatically finds and loads the correct media handlers and decompressors for the movie. An error is returned if these operations fail, but otherwise they are transparent to the application.

Nearly all QuickTime programmers need to deal directly with components from time to time, however. For example, to play a QuickTime movie, applications may work directly with a movie controller component.

Selecting and working with components is documented in *Component Manager for QuickTime* This is a subset of the *Component Manager Reference* for the Mac OS (Carbon); it describes the parts of the Component Manager that QuickTime programmers are likely to use, and that are included in QuickTime for Windows.

Specific QuickTime components are discussed in the QuickTime documentation for the topic relevant to that component. For example, movie controller components are described in *QuickTime Movie Playback Programming Guide* and compressor components are described in Compression and Decompression.

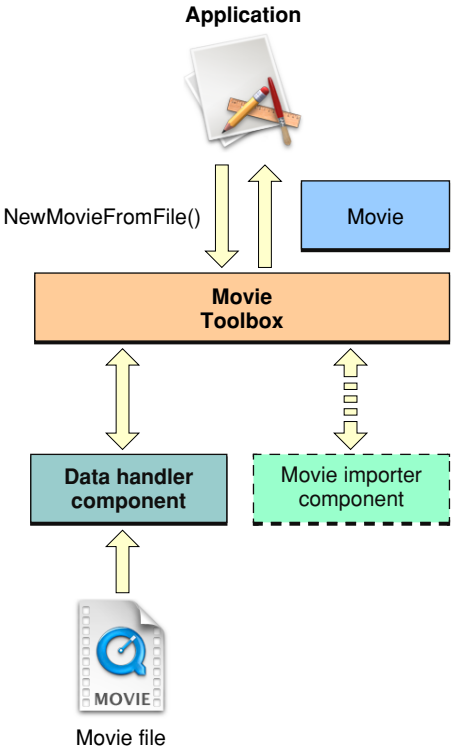
Note that the primary QuickTime documentation describes the interface to components from the perspective of an application calling the component. There is an additional set of documents that describe writing new components (see Creating QuickTime Components). If you are writing a QuickTime component, you need

to read the primary documentation for that component, the generic documentation for creating QuickTime components, and any additional documentation for creating that particular kind of component in order to understand how the component is used, what API you need to support, and how best to implement it.

Examples

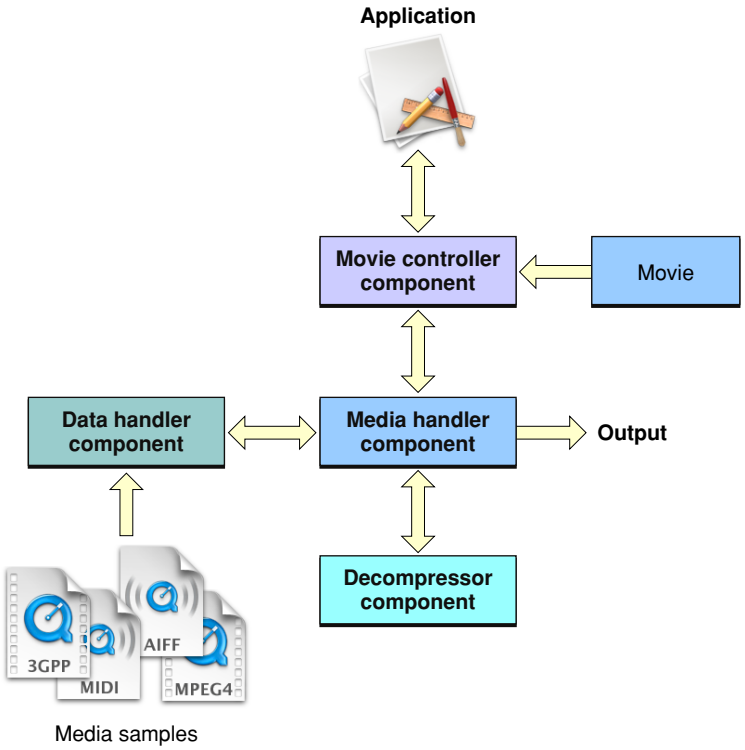
Let's look at two examples of the QuickTime architecture in action: getting a movie from a file and playing a movie using a movie controller.

Figure 1-3 Getting a Movie



- Your application tells the Movie Toolbox to get a movie, in this case from a file.
- A data handler component is used to access the file; a different component would be used to get a movie from a local file, a URL, or a stream. QuickTime chooses the appropriate component based on the data source.
- If the file is not a QuickTime movie file, a movie importer component is used to create a movie from the file; a different component would be used to import from an MP3 file or a JPEG file. QuickTime chooses the appropriate importer based on the file type, file extension, or MIME type.
- The Movie Toolbox passes a movie to your application.

Figure 1-4 Playing a Movie



- Your application attaches a movie controller to the movie and the user presses the “Play” button on the control bar.
- A media handler component is used to work with each type of media used in the movie; different components are used for sound, video, Flash, and so on. QuickTime chooses the appropriate media handlers based on the media types.
- Media handlers makes calls to data handlers to access their media samples, which may come from a different data sources. For example, a movie on a local disk might point to media samples on a local disk, a remote file server, and an Internet stream. QuickTime chooses the appropriate data handler for each data source.
- A media handler typically makes calls to a decompressor component to decompress the media samples; different components are used for different media types, such as sound and video, and for different compression schemes, such as MP3 or MP4 audio, JPEG or GIF images. QuickTime chooses a decompressor based on the media type and compression scheme.
- A media handler may then pass its output directly to a low-level service, such as the Sound Manager, for final disposal, or to another component, such as a video output component, for further processing.

Output

QuickTime can provide various kinds of output during playback. The output is typically sound and video, with a visible controller, but it can be simply sound, with no display or visible controller, or even a silent and invisible DV stream to a FireWire port.

The actual output is handled by a lower-level technology, such as Core Image, OpenGL, Core Audio, DirectX, or the Sound Manager. QuickTime shields you from having to deal with these details in most cases. When you play a movie, QuickTime selects and configures the default devices for playback on your platform.

If you need to work with QuickTime at a lower level, to modify individual video frames during playback, for example, or to add a filter to the sound output, you need to work with the underlying technology that QuickTime relies on for output. This will vary depending on your platform and software revision.

For example, QuickTime 7 for Mac OS X uses Core Audio for audio output. To work with QuickTime audio at the low level, you should use the Core Audio API. Older versions of QuickTime send audio output to the Sound Manager (included in QuickTime for Windows). To work with low level audio on these systems, you need to use the Sound Manager API.

By default the visual output from QuickTime 7 for Mac OS X goes to a graphics context managed by Quartz, while visual output on QuickTime for Windows and older versions of QuickTime for the Mac OS goes to a graphics port associated with a window, represented by an offscreen buffer called a `GWorld`. QuickTime handles decompression and visual rendering automatically.

QuickTime can be specifically directed to send its visual output to any device that has an installed video output component.

In addition, in QuickTime 7 and later you can create a visual context to specify a particular output format, such as Core Image pixel buffers or OpenGL textures. If you do this, however, you become responsible for rendering the decompressed frames to the screen. You can render the images using one of the underlying graphics Mac OS APIs, such as Core Video and Core Image, OpenGL, or QuickDraw. On Windows, you might use native Windows video APIs, OpenGL, or the parts of QuickDraw included in QuickTime for Windows.

Note: QuickTime programmers working on Windows OS or versions of the Mac OS prior to 10.4 need to learn a little about graphics worlds and QuickDraw to learn how to properly set up a graphics world, associate it with a graphics port, and work with the common QuickDraw data types, such as a rect. (See Color QuickDraw, Graphic Devices and *QuickDraw Reference*.)

Audio output goes to your system's default audio device by default, but QuickTime 7 and later support an audio context that allows you to specify any output device. QuickTime 7 and later also include some functions for working with audio, such as setting track volume, balance, and channel layout, and monitoring frequency levels during playback. To do sound processing or filtering, however, you need to use a lower-level technology such as Core Audio or DirectSound.

The QuickTime API

The QuickTime API allows you to add a host of multimedia features to your application without needing to master the often arcane details of particular media formats and specifications.

For example, you can use QuickTime to open and display a series of JPEG images, concatenate them into a time-based slideshow with an MP3 sound track, then export the images as TIFF or PNG graphics, or export the slides and music together as a DV stream, without needing to work directly with, or necessarily understand, the compression, stream, or file formats for JPEG, TIFF, PNG, MP3, or DV.

Multilevel API

The QuickTime API includes over 2500 functions, divided into tool sets for particular tasks, with special functions and data types for virtually any task.

This can be a little daunting for programmers new to the API. It is easy to get lost in details and lose sight of the big picture. The most common error among new QuickTime programmers is to attack a problem using a complex, low-level tool set when there is a much simpler high-level command to perform the entire task.

You can interact with the QuickTime API at many different levels:

- You can simply open and play movies, letting QuickTime handle all the file and format conversion, synchronization, data buffering, component loading and unloading, memory management, and even the user interface. Prebuilt controls are available for play/pause, volume control, time scrubbing, and cut-and-paste editing.
- You can control the playback or editing yourself, setting the play rate, scaling the duration of movie or track segments, rearranging the playback order, and so on, creating your own user interface and controls.
- You can work with individual components, loading particular importers or image decompressors, applying them to groups of files or blocks of memory, and disposing of them when you are done.
- You can work with individual data samples—synthesizing graphics and overlaying them on video frames as a movie plays, for example—or performing pattern recognition on groups of samples, or even generating whole movies programatically.
- You can write new QuickTime components to support features such as new compression algorithms, new media types, new media capture devices, output devices, or data sources.

Frameworks

You can work directly with the QuickTime API using C, C++, Objective C, or Java. There are also QuickTime interfaces for JavaScript, and Windows interfaces for Visual Basic, C#, and other COM or .Net frameworks. Other languages, such as BASIC or Pascal, may support indirect calls to the QuickTime API through calls to a directly supported language or framework.

Most developers call the QuickTime API from procedural C programs written in C or C++, and most of the QuickTime documentation describes this use of the API. Nearly all of the procedural C QuickTime API is also part of the Carbon framework for Mac OS X. However, the QuickTime API does include some Windows-only functions and some legacy functions from earlier versions of the Mac OS. The documentation of any function in the *QuickTime API Reference* tells you whether that particular function is included in Carbon.

The procedural C QuickTime API for Windows is almost identical to the Mac OS version, with a few exceptions and some minor modifications to avoid naming conflicts with the Windows operating system. These exceptions and modifications are documented in QuickTime for Windows Programmers. Some newer parts of the QuickTime API make use of advanced features of Mac OS X that are not available in Windows. In these cases the documentation will note an alternative API to achieve similar functionality in Windows, if this is possible.

There is also a QuickTime API for Objective-C, which corresponds to the Cocoa programming framework for Mac OS X. This rapidly growing part of the API is greatly streamlined and simplified. It is documented in QuickTime for Cocoa Programmers. It is also possible to call the procedural C parts of the QuickTime API from Cocoa programs if lower-level access to QuickTime is needed.

The Java API for QuickTime is documented in QuickTime for Java. You can make calls to the procedural C parts of the QuickTime API from Java.

Documentation of the JavaScript API can be found in the *JavaScript Scripting Guide for QuickTime*.

QuickTime Movies

When working with the QuickTime API, nearly all operations are performed on a data structure known as a QuickTime **movie.** The QuickTime movie is a description of a multimedia presentation. It tells a computer (or other multimedia-capable device):

- What type of media to present
- Where the data is located
- When and how to present each sample
- How to layer, arrange, and composite multiple elements

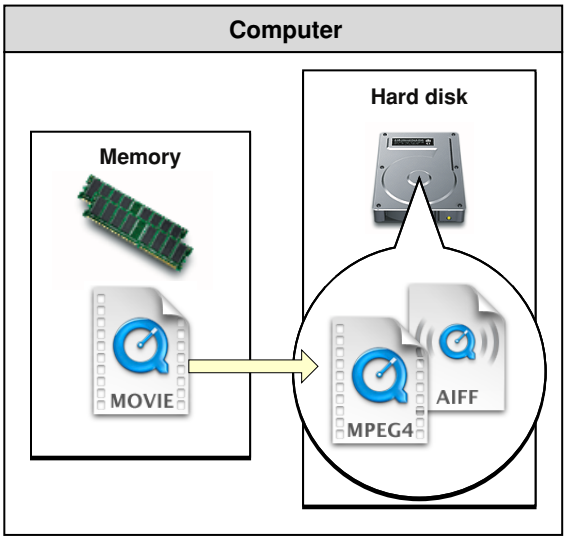
You can use QuickTime movies in several ways:

- Record digital data into QuickTime movies
- Import media from other formats into QuickTime movies
- Export from QuickTime movies to other formats
- Create, edit, and play all types of media (as QuickTime movies)

For example, to play an MP3 audio file using QuickTime, you create a new movie in memory from the MP3 file and play the movie. This does not directly copy the MP3 audio data into memory; it creates a small movie data structure that allows QuickTime to find, decompress, and play the audio data in the MP3 file.

A QuickTime movie does not contain sample data, such as audio samples or video frames. A movie is the organizing principle that allows a computer to locate and interpret the required sample data. Playing a movie causes QuickTime to locate and obtain sample data from wherever it is, decompress and composite it as necessary, and present it in the proper sequence and arrangement.

Figure 1-5 Sample data resides outside of the movie



High-level QuickTime operations, such as opening and playing movies, can often be performed with no need to delve into any details of a particular movie, such as what kind of media are presented, how the media are compressed, or where the data samples are stored. Still, a basic understanding of QuickTime movie structure is useful for any QuickTime programmer and is essential for lower-level operations.

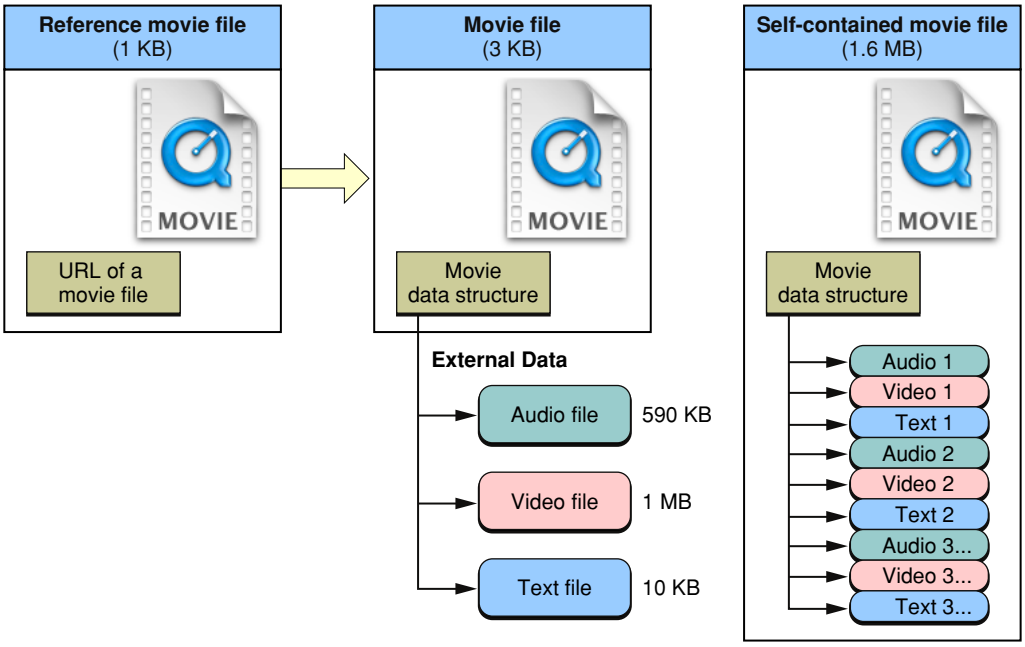
Movies and Movie Files

It's important to distinguish between a QuickTime movie, the data structure we have been discussing, and a QuickTime **movie file.** A movie is not the same as a movie file.

A movie file can contain a stored copy of a movie data structure, or it can contain only a reference to such a structure, stored somewhere else.

If a movie file contains a stored movie, it can optionally contain the sample data used by the movie as well. This is sometimes called a self-contained movie file, and it is quite common. When the sample data is stored in a movie file, it is interleaved for smooth playback.

Figure 1-6 Three kinds of movie files



In casual use, a QuickTime movie file is sometimes simply called a movie. Similarly, a reference movie file may be called a reference movie, and a self-contained movie file may be called a self-contained movie. But in the QuickTime API documentation, the word "movie" always refers to a movie *data structure,* not a movie file. It is sometimes useful to think of a movie in memory as an instance of a movie stored in a file.

The copy of a movie stored in a movie file is sometimes referred to as a **movie resource** to distinguish it from a movie in memory.

Note: In early versions of QuickTime, the movie data structure was stored in the resource fork of Mac OS files; hence the name "movie resource." This is no longer the case, but the name remains, and the distinction is sometimes useful.

Tracks

A QuickTime movie is organized into **tracks.** A movie can contain many tracks; there are practical limits, which change as computers become more powerful, but there is no predefined limit.

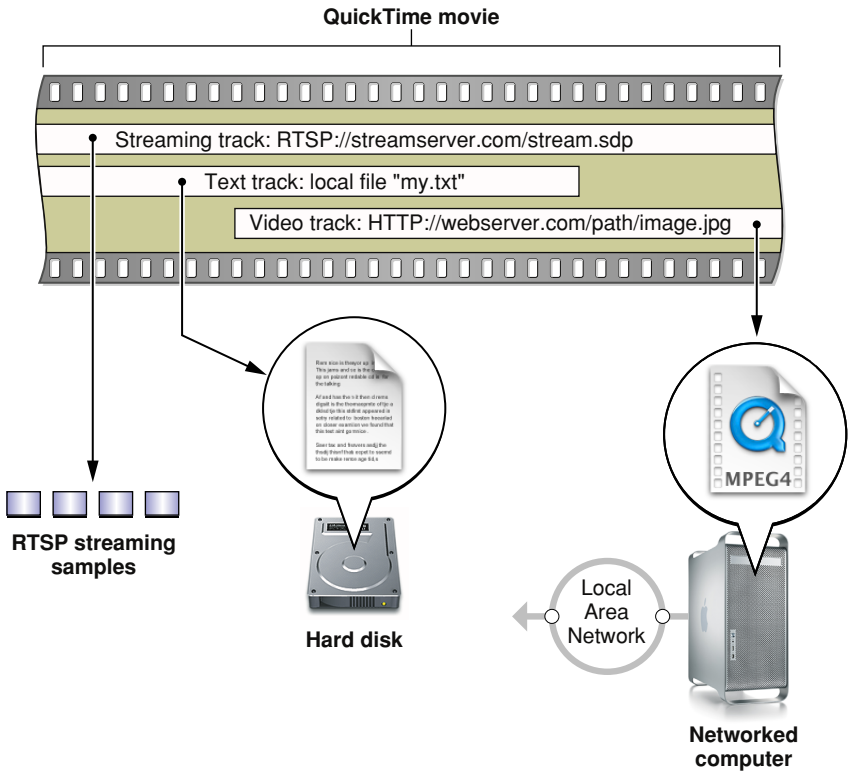
Track Media, Compression, and Data References

Each track specifies a **media type—such** as video, sound, or text—and a **data reference** that specifies where the sample data for that track can be found. A track may also specify a **compression format** (such as JPEG video or GSM audio).

The data reference may be to a local file, a file on a network or Internet server, a data stream from a network or Internet server, or a handle or pointer to a block of memory; other data reference types are also possible and the type itself is extensible. Simply put, the movie data can be anywhere. A data reference identifies the data source.

Different tracks can specify the same data source or different data sources. All the movie's media samples can be in a single file, for example, or the samples for a movie's sound track can be in one file while the samples for the video track are stored in a different file.

Figure 1-7 Movies can use data from multiple sources



A given track can specify only one media type, and most tracks get all of their samples from a single source. Some media types support multiple sources, however. For example, a video track can consist a series of JPEG images, each stored in a separate file. In this case, there is a data reference for every image.

Different tracks can be of the same media type or of different media types—you can have multiple video tracks and multiple sound tracks in the same movie, for example, or multiple text tracks in different languages.

A given track can use only one type of compression, but multiple tracks of the same media type may be compressed differently in the same movie. For example, a single movie can contain both MP3 and MPEG-4 compressed audio tracks.

Track Visual and Sound Characteristics

Visual tracks have properties such as height, width, x and y offsets, layer numbers, and graphics modes. This allows you to play multiple visual tracks at the same time: side by side, partly or completely overlapping, and with various degrees of transparency or translucence. Visual tracks also contain a transformation matrix that can be used for rotating, scaling, or skewing the visual output of a track at runtime. QuickTime provides automatic clipping of images at the track boundary, and can have an associated mask, or matte, for cropping the output using arbitrary shapes.

Sound tracks have properties such as volume and balance, allowing you to create layers of sound. Multichannel sound formats, such as four-speaker and 5.1 surround sound, are supported in QuickTime 7 and later.

Track Media

A track also contains a data structure known as a media. This is a low-level data structure that describes the location, duration, and natural time scale of the media sample data. This can be confusing, because in casual use the sample data itself is sometimes referred to as the track's media.

Important: Try not to confuse the media data structure with the data samples themselves.

When a QuickTime function or data type specifies a `media` as a parameter or field, it always refers to the media data structure inside a movie, not to actual data samples.

Media Time Scale

A QuickTime movie always has a **time scale,** expressed in units per second. You can specify a time scale when you create a movie, but the time scale cannot be changed once a movie exists. When you perform operations on a QuickTime movie, you frequently need to specify a point in the movie timeline at which to begin the operation; this is specified using a time value, expressed in movie time scale units. You may also need to specify a duration; this is also expressed in movie time scale units.

The default movie time scale is 600, so to advance a movie to a point 2 seconds into its duration, you would typically go to time 1200. Similarly, a duration of 1/30th of a second would be 20 time scale units. You can obtain the movie time scale by calling `GetMovieTimeScale`.

Tracks use the time scale of their parent movie. Time values and durations for all track operations are expressed in movie time scale units.

Each track's media, however, has its own time scale, which is typically the sample rate of the track's media data. For example, a track containing NTSC video might have a time scale of 30, while a track containing PAL video would have a time scale of 25, and a track containing CD audio would have a time scale of 44100. This allows you to conveniently refer to individual media data samples, increment through a group of samples, and so on.

Operations on individual media samples typically use times and durations expressed in the media time scale. You can obtain the media time scale for a given track by calling `GetMediaTimeScale`.

There are utility functions for translating between track time (which is also movie time) and media time. There are also numerous functions that allow you to translate between the time domain (time and duration) and the sample domain (sample number and number of samples).

Track Edit List

Each track contains an edit list, which allows you to alter or reorder the display of media samples without changing or rearranging the samples themselves. This results in nondestructive editing. You can "edit out" a track segment without deleting any samples from the data source, or repeat a segment without increasing the size of the data source with duplicate samples.

You can also use the edit list to alter the duration of a media segment, causing it to play back faster or more slowly than it normally would, or insert an "empty" track segment that contains no data for a period of time. In other words, any segment of media time, including an empty segment, can be mapped to any segment of track time.

If a track has not been edited, the edit list is empty and the track is treated as a single segment, with all the media samples played in the order they are stored, at the natural time scale for the media sample data.

Track Duration

Each track has a duration, which is the combined duration of all segments in its edit list (typically the combined duration of all of its samples), including any "empty" segments.

Similarly, each movie has a duration, which is simply the duration of its longest track.

Note: Technically, it is possible to offset the beginning of a track from the beginning of a movie by a fixed amount of time. In practice, this is rarely done; instead, an "empty" segment is inserted at the beginning of the track's edit list, so that the first sample is displayed after a fixed amount of time.

Samples

At the lowest level, a QuickTime track contains a set of sample tables. Each entry in a sample table specifies the location and duration of a chunk of sample data, such as a still image, a video frame, a sequence of PCM audio samples, or a text string.

There is at least one sample description for each table of samples. The sample description provides the details necessary to translate a stored sample into a format that the media handler can work with. For example, a sample description might specify the height, width, and pixel format of an image, or the sample size and sampling rate of a group of PCM audio samples.

For some media types, such as sound, all data samples in a given track share a single sample description. If you have audio samples that use different sample rates or sample sizes, for example, they must be in separate sound tracks.

Other media types can have multiple sample descriptions, so a series of images could have varying heights and widths, with different sample descriptions used whenever the dimensions change.

Sample Duration and Frame Rate

Because each chunk of sample data has its own duration, and a chunk can be as small as a single sample, a QuickTime track may not have any fixed "frame rate." A video track might consist of a series of images that act as a slideshow, for example, with each "slide" on screen for a different length of time.

This can be very difficult to grasp if you are used to working in media with fixed frame rates, but it is a powerful feature of QuickTime. A fixed frame rate would require images to be repeated periodically, perhaps many times, to display them on screen for an extended period; in QuickTime, each image can be stored as a single sample with its own unique duration.

By extension, a QuickTime movie does not necessarily have a fixed frame rate. A 25-fps PAL video track may play side by side with a 30-fps NTSC video track in the same movie, for example, perhaps with both tracks composited on top of a still image that is displayed for the entire duration of the movie, or on top of a "slideshow" track that changes at irregular intervals. This is possible because the display is created at runtime by a programmable device, not mechanically projected by display hardware.

Of course, a QuickTime track, or a QuickTime movie, may have a frame rate; it is very common for a video track to contain a series of samples that all have the same duration, and it is also common for a movie to have a single video track with a constant sample rate. But it is not a requirement.

You can always compute a frame rate by dividing the duration of a track by the total number of video samples, but be aware that the results of this calculation are not always predictive of the movie's behavior; the actual frame rate could change abruptly at several points during the movie.

Time

As noted in the discussion of tracks, a movie has a **time scale,** as does the media for each track. A time scale specifies some number of units per second. For a media, the time scale is usually the sample rate. For a track or a movie, the time scale can be any convenient number (the track time scale is the same as the movie time scale).

Note: The default time scale for a movie is 600 units per second. You can specify a time scale when you create a movie or add a blank media to a new track.

The relationship among the movie's time scale and the time scales of the various media define the movie's **time-coordinate system.** QuickTime uses the movie's time-coordinate system to synchronize all the tracks and media to the movie timeline.

A movie always has a **current time,** which designates what parts of the movie should be presented immediately. The current time is expressed in movie time-scale units. For example, if the movie time-scale is 600, and the movie has been playing for half a second, the current time is 300.

The current time can range from 0 to the movie's duration. Current time changes as the movie plays. Dragging the playhead in a movie controller changes the current time in the movie.

A movie also has a **rate,** which is 0 when the movie is stopped and 1 when the movie is playing at its normal speed, which is defined by the movie time-scale. For example, a movie with a time-scale of 600 plays at 600 units per second when the rate is 1. Negative rates cause the movie to play backward. Rates greater or less than 1 cause the movie to play faster or slower than normal. For example, a movie with a time-scale of 600 plays at 300 units per second when the rate is 0.5, and at 1200 units per second when the rate is 2.

QuickTime establishes a playback **time base** when a movie is run. The time base consists of the movie's time-coordinate system, a rate, a current time, and a reference to a **clock component** that provides QuickTime with measurements of real time. This allows QuickTime to play a movie at the correct number of time-scale units per second for the current rate in real time.

This also allows QuickTime to "drop frames" appropriately if the data rate of the movie exceeds the capability of the playback device, so tracks remain synchronized with each other and with real time specifications (for example, a one-minute movie plays in exactly one minute, even if the playback device cannot decompress all of the movie's video frames in that length of time).

Linear and Nonlinear Media and Movies

Linear media, such as a series of consecutive video frames, are tied to the movie timeline; they change in a fixed manner as the current time changes, varying in tempo and direction with the movie's rate.

QuickTime also supports **nonlinear** media, such as a bouncing sprite, whose actions can be specified with respect to the passage of real time, or with respect to user actions such as mouse clicks. These actions can continue even when the movie is paused (has a rate of 0). This makes it possible to embed customized controls in a movie that respond to user interaction.

Movies that normally play at a fixed rate are called linear movies and typically feature a controller with a play/pause button and a time-slider. Movies that are nonlinear may feature a different type of controller or no controller at all.

For example, a VR panorama is usually controlled by a special VR controller that changes the image in response to the keyboard and mouse. A VR movie normally has a rate of 0, because it consists of a still image that the user can interact with. The VR image is nonlinear; it does not change in a fixed manner during the movie timeline, but in response to unpredictable user actions.

Nonlinear movies can use the movie timeline to separate distinct behaviors. For example, if a panorama has multiple nodes, each node is located at a different point on the movie timeline to keep them from displaying simultaneously; jumping to a new node involves changing the current time, typically while leaving the rate at 0.

It is possible to mix linear and nonlinear media in the same movie. To add sound to a VR panorama, for example, the duration of the VR image is extended to match the duration of the sound track. When the rate is nonzero, the sound plays. The display of the panorama remains nonlinear, however; it changes when the user interacts with it, without regard to the movie's current time or rate (as long as the current movie time is within the VR image's duration). If the movie is paused, for example, the sound stops playing but the VR image remains interactive.

When mixing linear and nonlinear media, it is sometimes necessary to create custom movie controls. For example, the VR controller has no play/pause button to start and stop a sound track. You can control the movie rate programatically from your application or add an interactive sprite to the movie, such as a play/pause button, to provide user control.

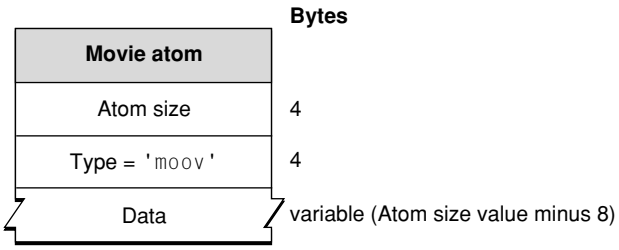
Atoms, QuickTime Atoms, and Atom Containers

QuickTime makes frequent use of data types known as **atoms.** You do not normally need to deal with QuickTime at the atomic level. In general, there are higher-level functions that allow you to, for example, create a movie, add or delete a track, or set a track's media type, without directly manipulating, or necessarily knowing anything about, atoms.

To understand the various ways movies can be delivered over a network, however, it is useful to know how movies are stored in files, and QuickTime movies are stored in files as atoms.

An atom is simply a container; it has a 4-byte **length** field, which specifies its total size (including the length field), and a 4-byte **type** field, typically four ASCII characters, which specifies the type of atom it is. The type field can be followed by data, the amount and kind of data depending on the atom type.

Figure 1-8 Atom layout



The smallest possible atom is therefore 8 bytes: a 4-byte size field and a 4-byte type field, with no data.

Note: Because no atom can be smaller than 8 bytes, the size field can contain the values 0 through 7 as special flags; a flag could indicate, for example, that the atom extends to the end of a file, or that the atom is larger than a 32-bit field can describe, and the actual size is in a 64-bit field following the type field.

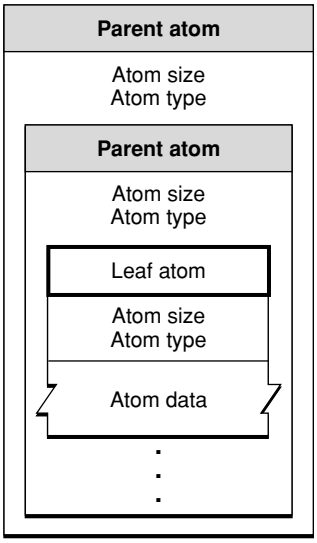
One atom can contain other atoms, allowing you to nest them in arbitrary hierarchies. This makes atoms handy building blocks for larger data structures.

You can "insert" one atom into another simply by appending the new atom and adding its size to the size field of the original atom. The original atom has now been extended to "contain" the new atom. If the original atom is inside yet another atom, that atom can be extended in the same manner.

An atom inside another atom is sometimes called a **child atom.** Child atoms at the same level in a hierarchy are called **siblings.** An atom that has other atoms inside is called a **parent atom** or **container atom.** An atom that has tabular data inside, instead of other atoms, is called a **leaf atom.**

Note: It is possible for an atom to contain both tabular data and other atoms, but this is discouraged.

Figure 1-9 Parent atoms and leaf atoms



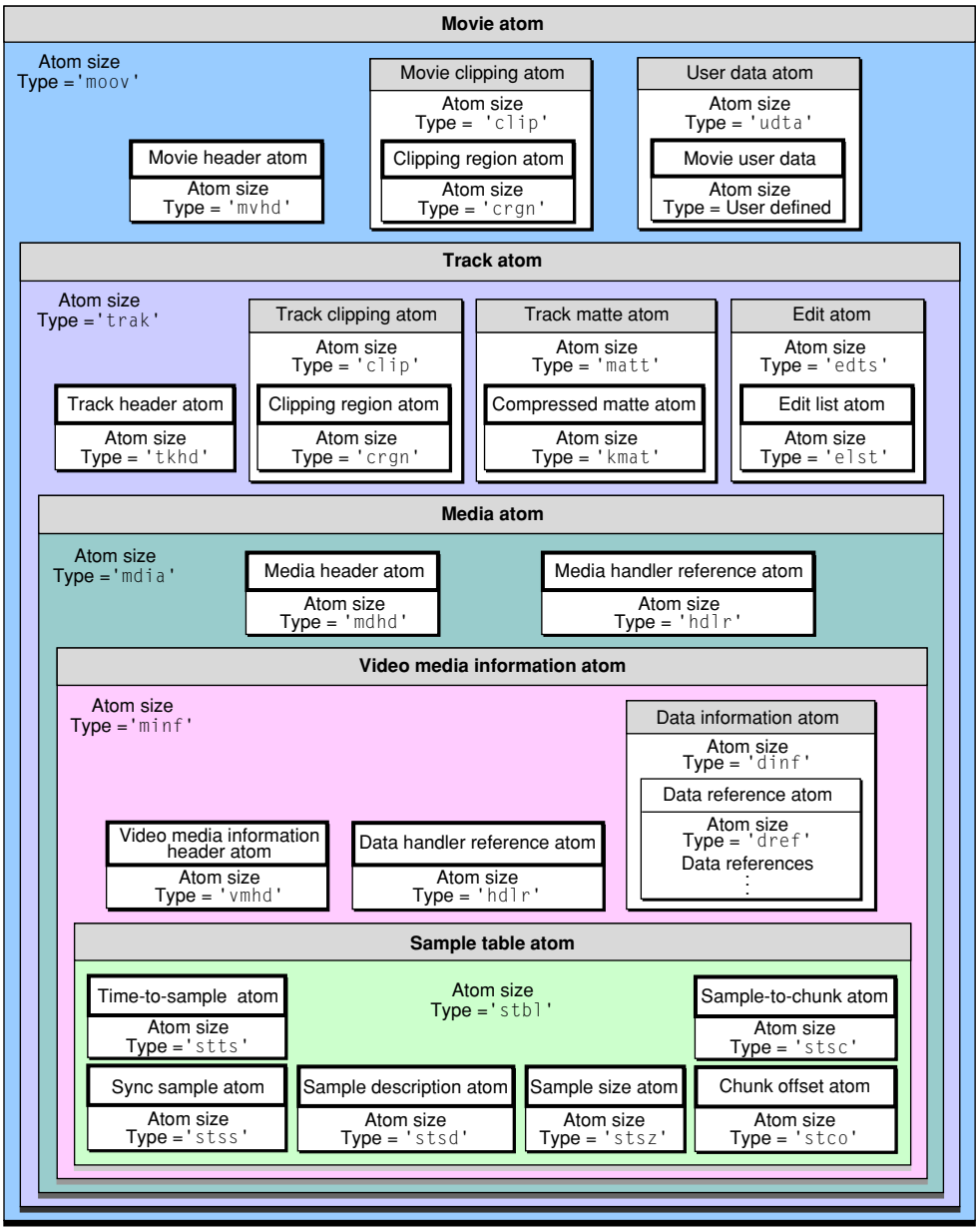
Because an atom begins with its size, it is easy to "walk" an atom structure by skipping from atom to atom within the structure. You can quickly scan a collection of atoms for an atom of a particular type or skip over an atom you are not interested in.

A QuickTime movie is a parent atom whose type is `'moov'`.

Each track in a movie is a child atom of type `'trak'` inside a `'moov'` atom.

Each track atom contains other child atoms, such as an edit list atom and a media atom. These atoms in turn contain other atoms, such as a media handler atom or various sample table atoms. The sample tables are leaf atoms.

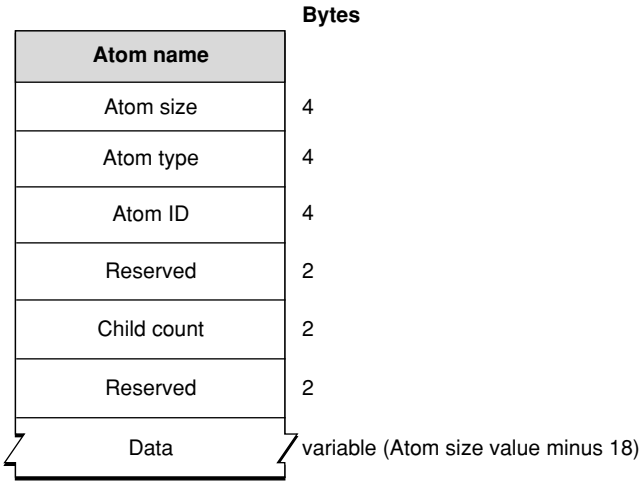
Figure 1-10 Layout of a movie atom



Specific atoms are documented in the *QuickTime API Reference*, and may also be described conceptually in the documentation of a relevant topic, function, or group of functions. Additional details of the content and structure of various atoms can be found in the QuickTime File Format specification.

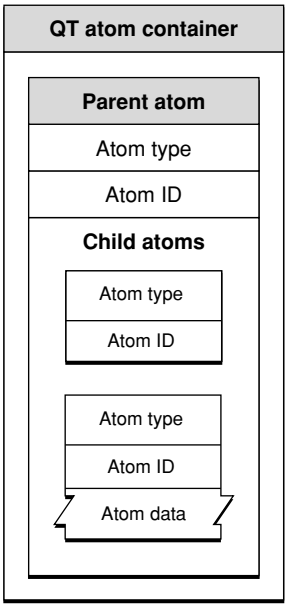
A refinement of the basic atom, used for some types of QuickTime data, is the QuickTime atom, or **QT atom.** This type of atom has additional header fields that specify the version of a particular atom type, an ID that allows you to distinguish one atom of a given type from its siblings in a hierarchy, and other useful information. It also has the restriction of containing either tabular data or other atoms, but never both. (If a QT atom needs to contain data about itself, in addition to containing other atoms, the necessary data is simply wrapped in an atom.)

Figure 1-11 QT Atom Layout



At the highest level, a hierarchy of QT atoms is stored in an **atom container.** This is a unique data structure (not an atom) that contains a hierarchy of QT atoms.

Figure 1-12 Atom container and QT atoms



Important: An atom container is *not* an atom; it is a non-atom data structure that *contains* a hierarchy of QT atoms. Do not confuse it with a "container atom" (another name for a parent atom), which is simply an atom that contains other atoms. To prevent confusion with atom containers, the term "parent atom" is preferred rather than "container atom."

Complex data structures, such as compression settings, are commonly stored in atom containers. This allows these data structures to be flexible and extensible, unlike a rigid struct. Properties can be stored as atoms, and it is possible to find out if a particular instance of the structure has a given property, and what data format the property takes, before getting or setting the property. The QT atoms within a container are typically accessed by their byte offsets within the container, which makes data transfer quick and efficient.

Streaming, Broadcasting, and Progressive Download

QuickTime movies are usually stored to disk in QuickTime movie files. These files often contain the sample data used by the movie as well. The QuickTime API includes functions to store a movie, or a movie and all its associated sample data, to a file. By default, the movie data structure is stored at the beginning of the file (in the form of a `'moov'` atom) followed by any sample data (typically wrapped in an `'mdat'` atom). By default, the sample data is interleaved, so that media samples that are displayed at the same time are stored close together, with the samples needed earliest stored first.

This typical movie file can be delivered by any web server, using common protocols such as HTTP and FTP, just as if it were an HTML file or a JPEG image. It is necessary only to name the file correctly and associate the filename extension with the correct MIME type on the server. (The correct filename extension for QuickTime movies is .mov, and the correct MIME type is `'video/quicktime'`.)

When a file is delivered over a network or downloaded over the Internet, the entire file is not available immediately, but a typical QuickTime movie can be played while it downloads. This is called **progressive download,** or **Fast Start.** It works because the movie atom is stored at the beginning of the file, so QuickTime knows how to interpret the movie sample data even before it arrives, and because the movie data is intelligently interleaved with respect to display time.

It is also possible to create a movie file with the sample data stored first, followed by the movie data structure. This is not usually desirable, because the entire file must download before QuickTime can interpret the sample data. You can correct this kind of data inversion simply by opening the movie file in QuickTime and saving it as a new, self-contained file. QuickTime stores the movie data structure at the beginning of the file by default.

A QuickTime movie file may contain *only* a movie data structure, pointing to sample data in other files or URLs. In most cases, this type of movie can also play as the movie data downloads, because, again, the movie data structure allows QuickTime to interpret the incoming data, and because the data source for each track is specified independently, causing the network to perform a kind of interleaving by delivering all of the media independently and simultaneously. Obviously, this kind of interleaving is less reliable than the deliberate interleaving QuickTime does when creating a self-contained movie file, so playback may not always be as smooth.

When the bandwidth of a connection meets or exceeds the data rate of the movie, a well-formed QuickTime movie file can play as it downloads. This kind of progressive download, or Fast Start movie, provides the same user experience as real-time streaming.

If the connection is not fast enough to play the movie in real time, you can either wait until the download completes or play as much of the movie as has downloaded at a given time. QuickTime can even estimate the required download time and begin playback when it calculates that enough data has arrived to play the movie smoothly (because the remaining data is expected to arrive by the time it is needed).

QuickTime movies can also be delivered using real-time protocols such as RTP and RTSP. This requires a streaming server, such as the QuickTime Streaming Server or Darwin Streaming Server. To stream movies in real time, the server requires information about how to packetize each track in the movie. This information is stored in special tracks in a QuickTime movie, known as **hint tracks.** There are functions in the API for adding hint tracks to existing movies, as well as flags that can be used to tell QuickTime to create hint tracks when saving a movie to disk.

Movies with hint tracks can also be delivered using HTTP or FTP protocols for progressive download, but additional bandwidth is needed to carry the hint tracks, which are used only for streaming. Consequently, it is best to determine how you will deliver a movie before saving it as hinted or nonhinted.

In addition to progressive download and real-time streaming of stored movie files, QuickTime supports **broadcasting,** the creation of one or more real-time streams from real-time sources, such as cameras or microphones. This involves capturing the incoming data, compressing it to the desired bandwidth, and generating streams of outgoing packets, all in real time. The QuickTime broadcast API is currently available for the Mac OS only; it is not available for Windows or Java.

Streaming Versus Progressive Download

There are trade-offs to consider when deciding whether to deliver a movie using progressive download, streaming, or broadcasting.

All QuickTime media types can be delivered as progressive downloads. Streaming is limited to sound, video, and text. Broadcasting is further limited to compression schemes and quality settings compatible with real-time capture and compression.

Progressive download works even when the bandwidth is not sufficient for real-time playback; it simply buffers incoming data and delivers delayed playback. Streaming and broadcasting are bandwidth limited; if the connection is not fast enough, the movie cannot play.

Streaming movies do not store a copy of the movie on the client computer, making them inherently more difficult to copy without the consent of the movie's owner. This can be an important consideration, and is one reason why people choose streaming over progressive download.

Streams take up a specified amount of bandwidth, whereas HTTP file downloads proceed as quickly as the connection allows. It is therefore easier to manage the bandwidth usage of a streaming server than of a web server delivering progressive-download movies.

Broadcasting allows you to deliver coverage of live events as they happen, or to provide real time "chat" between computers.

To sum up, if your movie includes live coverage, you must use broadcasting. If bandwidth management and copy discouragement are paramount considerations, streaming may be your best choice for stored content. If simplicity, reliability, or quality regardless of connection speed are most important to you, progressive download is probably best.

Combined Movies and Reference Movies

It is possible to combine progressive download media with streaming or broadcast media in a single QuickTime movie, providing the best features of each delivery method. This is done by creating a self-contained QuickTime movie for progressive download, then adding tracks whose data references specify the RTSP URL of a live broadcast or a hinted movie on a streaming server.

A QuickTime movie with a custom media skin and wired sprites can act as a customized movie player application that plays streaming media from a predefined source. This is a relatively common application for promoting music and music videos.

A QuickTime movie file may not always contain a movie data structure, at least not directly. It may contain a reference, such as a path and filename, or a URL, specifying another movie file. It can also contain references to several movie files, with specified criteria for choosing a particular file. These kinds of movies are called **reference movies.** For example, you can create a reference movie with three URLs, pointing to three versions of the same movie compressed at different bit rates, and specifying the preferred connection speed for each version.

A reference movie can refer to Fast Start movies, stored streaming movies, or live streams, so a reference movie can be used to improve the user experience with any kind of Internet delivery: broadcast, streaming, or progressive download.

QuickTime Road Map

QuickTime is a large API, with over 2000 functions and dozens of components. Fortunately, you normally need to use only a small part of the QuickTime API to accomplish a given task.

The trick is to know what part of QuickTime to use for your purpose, and to find the documentation and sample code that can guide you. That's what this roadmap is for.

QuickTime usually provides multiple ways to do the same thing: an easy way where most things are done for you, and a set of increasingly lower-level toolsets for doing it yourself. If you find that you're immersed in something complicated and frustrating, when it ought to be simple, there's a good chance that you're using the wrong toolset. Come back to the roadmap and look for a higher-level approach.

Main Areas of Interest

The QuickTime API documentation is divided into 17 main areas, listed and linked below.

- *Getting Started with QuickTime*—Getting oriented, finding the sdks, suggested reading
- Fundamentals— QuickTime overview, quick-start tutorial, component manager, initializing QuickTime, opening and playing movies
- QuickTime for Windows—Aspects of QuickTime that are different in Windows
- Scripting—Control Apple's QuickTime applications (player, browser plug-in, ActiveX control) with high-level scripting languages such as JavaScript, Visual Basic, AppleScript, HTML, and SMIL.
- Movie Basics—Initialize QuickTime, open and play movies, edit and save movies, work with QuickTime data types, set up callbacks to your application.
- Streaming—Work with streaming media using real-time protocols, do live broadcasts, or write components for the streaming server.
- Movie Internals—Work with movies at the track, property, and component level; set time scales and layers; rotate, skew, scale, and transform visual tracks; work with clock components, track references and modifier tracks, previews and media access keys.
- Movie Creation—Capture or synthesize data and create your own movies; use the sequence grabber, media-specific sequence grabber channel components (such as text), and video digitizers.
- QuickTime Import and Export—Bring existing media into QuickTime from dozens of other formats, export movies, tracks, or images to various non-QuickTime formats and file types.
- Compression and Decompression—Set up and work directly with image and sound compressors and decompressors, data codecs, and image transcoders.
- Video Effects and Transitions—Work with QuickTime filters, wipe and fade transitions, and other effects.
- Media Types and Media Handlers—Work directly with media handlers for video, sound, text, timecode, and more, including tween components.
- Wired Movies and Sprites—Animate sprites programmatically, add interactive controls and wired actions to movies, make movies that interact with remote servers.
- Virtual Reality—Work with QuickTime VR panoramas and cubes, set up hotspots, control cursors, add interactive features.
- Music and Audio—Work with digital sound at a low level, or use the QuickTime Music Architecture to synthesize music.
- Transport and Delivery—Use data handler components and video output components to get movies and media from special data sources (such as databases) or to send video to output devices other than screens.

- Writing Components—Extend QuickTime by writing your own components for new media types, compression schemes, data sources, output devices, clock sources, and more.

The main QuickTime documentation page contains links to all these areas. Clicking a link brings up a list of documents in that area; you can sort the documents by date, title, or topic. The areas are listed alphabetically.

Document Revision History

This table describes the changes to *QuickTime Overview*.

Date	Notes
2005-08-11	Revised to show that tracks use the movie time scale, and that track media have an independent media time scale.
2005-07-07	Updated for QuickTime 7.
2005-06-04	Restructured document to meet internal standards.
2004-12-02	Revised to correct links and minor textual errors.
2004-10-18	New document that provides a high-level overview of QuickTime.

Document Revision History